Final Project: Guestbook App

The whole series in one small app — a form, validation, saving to MySQL, a list page, and a flash message — built step by step and tested live.

Introduction

Twelve lessons of pieces — now one whole thing. The **Guestbook App** is a page where visitors sign their name and a message, and every visitor sees the list of everyone who signed before them. It is small enough to build in one sitting and real enough to use: a form (PH.9), validation (PH.9), a session flash message (PH.10), MySQL saving and reading (PH.11), and loops printing HTML (PH.5, PH.7).

Build it in four stages, in order. Stage 1: the database table. Stage 2: the form that saves. Stage 3: the list that reads. Stage 4: the flash message and the polish. Test after each stage — never write all four and hope.

(Roman Urdu: Ab sab kuch ikattha — form, jaanch, database, list, aur flash message. Aik chhota asal app, chaar stages mein.)

Key points

- **One project, four stages:** table → save form → list page → flash message and polish. Test after each stage.
- **The same five students power the examples** (Ali 88, Sara 92, Bilal 76, Hira 95, Usman 64) — the running cast of the whole series.
- **The form posts to the same page** (`action=""`), and the `REQUEST_METHOD` check separates "show the form" from "handle the submit".
- **Validation order is fixed:** read → trim → check empty → check length → save or show errors.
- **Saving always uses a prepared statement** (PH.11) — the guestbook is public, so anyone can type anything into it.
- **The flash message lives in `$_SESSION` for exactly one request** — set it, redirect, show it, delete it. That is the PRG pattern: Post → Redirect → Get.
- **`header("Location: ...")` must run before any output** — like `session_start()`, it sends headers, so no HTML may come first.
- **Every printed message passes through `htmlspecialchars`** — a guestbook is the classic place where an attacker tries to inject a script.
- **Order the list newest-first** with `ORDER BY id DESC` — visitors want to see the latest signature at the top.
- **When it works, the next step is Laravel** — the same app in Laravel is LV.6 (CRUD project), and you will recognise every piece.

Examples

Example 1 — Stage 1: the table

Run this once in phpMyAdmin (SQL tab, `school` database):

```
CREATE TABLE guestbook (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    message TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

What you will get:

```
A new empty table called guestbook, with 4 columns.
```

`AUTO_INCREMENT` numbers each signature for you. `created_at` stamps itself — you never insert it by hand.

Example 2 — Stage 2: the form that saves (guestbook.php, part 1)

```
<?php
session_start();
$conn = mysqli_connect("localhost", "root", "", "school");
$errors = [];
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = trim($_POST["name"] ?? "");
    $message = trim($_POST["message"] ?? "");
    if ($name == "") $errors[] = "Name is required.";
    if ($message == "") $errors[] = "Message is required.";
    if (strlen($message) > 500) $errors[] = "Message must be under 500 characters.";
    if (empty($errors)) {
        $stmt = mysqli_prepare($conn, "INSERT INTO guestbook (name, message) VALUES (?, ?)");
        mysqli_stmt_bind_param($stmt, "ss", $name, $message);
        mysqli_stmt_execute($stmt);
        $_SESSION["flash"] = "Thanks, $name! Your message is on the board.";
        header("Location: guestbook.php");
        exit;
    }
}
?>
```

What you will get:

```
A valid submit saves the row and jumps back to the same page.
An empty submit reloads the page with the errors list filled.
```

`$_POST["name"] ?? ""` reads the field or falls back to empty — no warnings on first load. `exit` after the redirect stops the page from printing the form underneath the redirect.

Example 3 — Stage 3: the list that reads (guestbook.php, part 2)

```
<?php
// ...continued from Example 2, same file, after the PHP above...
$result = mysqli_query($conn, "SELECT name, message, created_at FROM guestbook ORDER BY id
DESC");
$entries = [];
while ($row = mysqli_fetch_assoc($result)) {
    $entries[] = $row;
}
?>
<h1>Guestbook</h1>
<?php if (!empty($errors)): ?>
  <ul style="color:red">
    <?php foreach ($errors as $e): ?>
      <li><?= htmlspecialchars($e) ?></li>
    <?php endforeach; ?>
  </ul>
<?php endif; ?>
<form method="post" action="">
  <input type="text" name="name" placeholder="Your name"><br>
  <textarea name="message" placeholder="Your message"></textarea><br>
  <button>Sign the guestbook</button>
</form>
<h2>Signatures (<?= count($entries) ?>)</h2>
<?php foreach ($entries as $entry): ?>
  <p><strong><?= htmlspecialchars($entry["name"]) ?></strong>
  <small><?= htmlspecialchars($entry["created_at"]) ?></small><br>
  <?= htmlspecialchars($entry["message"]) ?></p>
<?php endforeach; ?>
```

What you will get:

```
A form at the top, and below it every signature, newest first,
each name escaped and safe to display.
```

`<?= ... ?>` is the short echo tag — it prints a value. The whole list is one `foreach` over the rows, exactly the pattern from PH.7 and PH.11.

Example 4 — Stage 4: the flash message (top of guestbook.php)

```
<?php
// ...right after session_start(), before anything else prints...
if (isset($_SESSION["flash"])) {
    $flash = $_SESSION["flash"];
    unset($_SESSION["flash"]);
}
?>
<?php if (!empty($flash)): ?>
  <p style="color:green"><?= htmlspecialchars($flash) ?></p>
<?php endif; ?>
```

What you will get:

```
After a valid submit, the page shows: Thanks, Ali! Your message is on the board.
Reload the page: the message is gone — it lived for exactly one request.
```

Read it, show it, delete it — that is the whole flash pattern. Because the save redirected first, a reload never re-saves the same message (the PRG pattern).

Example 5 — The full file, checked end to end

Put Examples 2, 3 and 4 together in one file, `htdocs/guestbook.php`, in this order: session start → flash read → form handling (validate, save, redirect) → fetch entries → HTML (flash, errors, form, list). Then test it like a user:

```
<?php
// Test checklist — do each one, in order:
// 1. Open guestbook.php: form shows, zero signatures.
// 2. Submit empty: two red errors, nothing saved.
// 3. Submit name "Ali", message "First!": green flash, 1 signature.
// 4. Reload the page: flash gone, signature still there, still 1.
// 5. Submit message "<script>alert(1)</script>": saved as text, shown as text, no alert.
// 6. Submit a 600-character message: length error, nothing saved.
?>
```

What you will get:

```
All six checks green = the app works. Any red check = go back to its stage.
```

A checklist is not optional decoration. It is how you prove the app works instead of believing it does.

Example 6 — Next step: the same app, one level up

```
<?php
// What you just built, in one line each:
// - A table with an auto id and a timestamp (PH.11)
// - A form with validation (PH.9)
// - A prepared statement that saves safely (PH.11)
// - A loop that prints rows as HTML (PH.5, PH.7)
// - A session flash message (PH.10)
// Laravel's LV.6 builds this same app with routes, a controller,
// Blade views and Eloquent — and now every one of those words
// maps to something you have already built by hand.
?>
```

What you will get:

```
A clear map of what you know — and what Laravel will rename, not reteach.
```

Quick recap

- Four stages: table → save form → list page → flash message and polish.
- Validate in order: read, trim, empty check, length check, then save or show errors.
- Save with a prepared statement; print with `htmlspecialchars`.
- Flash = set in session, redirect, show once, delete.
- Newest first with `ORDER BY id DESC`; test with a six-point checklist.

Common mistakes

1. **Printing HTML before the redirect** — `header("Location: ...")` after any output fails with "headers already sent". Fix: all handling PHP (session start, validation, redirect) runs before the first HTML tag.
2. **Forgetting `exit` after the redirect** — the page keeps running and prints the form underneath. Fix: `header(...); exit;` always together.
3. **Forgetting to delete the flash after showing it** — it haunts every later page. Fix: `unset($_SESSION["flash"])` the moment you read it.
4. **Printing messages without htmlspecialchars** — the guestbook is the first place an attacker tests. Fix: escape every printed value, every time.
5. **Skipping the redirect (no PRG)** — reloading the page re-saves the same message again and again. Fix: after a successful save, always redirect.
6. **Validating after saving** — the bad row is already in the database when the error shows. Fix: validate first, save only when `$errors` is empty.

7. **Testing only the happy path** — the app works for good input and breaks on the first empty submit. Fix: run all six checklist tests, including the empty and the 600-character ones.

Practice

1. Create the `guestbook` table from Example 1 in phpMyAdmin.
2. Build `guestbook.php` from Examples 2–4 and sign it with your own name.
3. Run all six checks from Example 5 and write down the result of each.
4. Add a 5th stage: a "Delete" link per entry that removes that row (use its id, and a prepared statement).
5. Add a search box that filters the list by name with `WHERE name LIKE ?` and a prepared statement.
6. Add a counter at the top: "N signatures so far" using `count($entries)`.
7. Show your guestbook to a friend and ask them to break it — empty submit, giant message, a `<script>` tag — and write down what held and what did not.

Practice quiz

1. What is the fixed order for handling a form in the guestbook?

- Save, then validate
- Read, trim, check empty, check length, then save or show errors
- Show errors first, then read
- Redirect first, validate later

Answer: Read, trim, check empty, check length, then save or show errors

2. Why redirect after a successful save (the PRG pattern)?

- To make the page slower
- So reloading the page does not save the same message again
- To delete the message
- Redirects are required by MySQL

Answer: So reloading the page does not save the same message again

3. How long does a flash message live?

- Forever, until the user deletes it
- Exactly one request: set it, redirect, show it, delete it
- One year
- Until the browser closes

Answer: Exactly one request: set it, redirect, show it, delete it

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/ph-final-project · More free tools and notes at https://scopetreasure.com