Unit 4: Form Handling and Data Validation

Presented By:
Bipin Maharjan

Learning Objective

- Gain proficiency in processing HTML forms to collect user input from web pages.
- Understand the HTTP request methods and their usage in form submissions.
- Validating and sanitizing user input to prevent security vulnerabilities and ensure data integrity.
- Validating file types, sizes, and other attributes to ensure the integrity of uploaded files and prevent potential security risks.
- Understand the process of sending emails programmatically.

Table of Contents

Processing HTML forms with PHP

What is a Form?

A form is a section of an HTML page where users can enter data (like name, email, password, or upload a file).

Forms are essential for interaction: login pages, search boxes, contact forms, registration pages, etc

Syntax

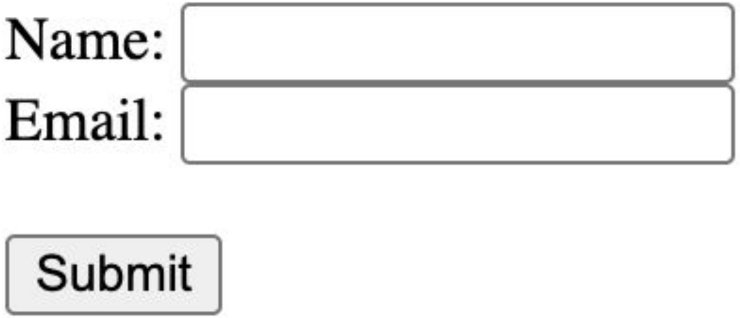
```
<form action=”PageToOpen” method=””>

//input controls placed here

</form>
```

Method=GET or POST

Basic Form Structure

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

    <label for="name">Name: </label>

    <input type="text" name="name" id="name" required>

    <br>

    <label for="email">Email: </label>

    <input type="email" name="email" id="email"
required>

    <br><br>

    <input type="submit" value="Submit">

</form>
```

Output



Form Attributes

- action="process.php" → The PHP file that will handle the form data.
- method="post" → The way data is sent to the server (GET or POST).

PHP Superglobals for Form Data

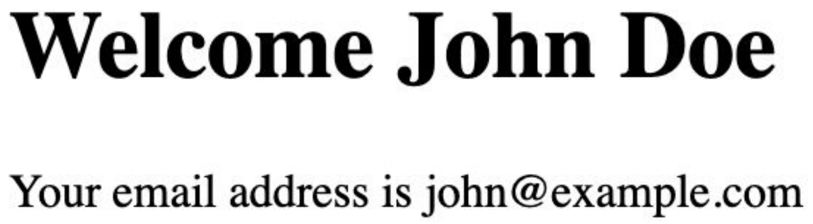
When the form is submitted, PHP automatically makes the data available through superglobal arrays:

- $_POST → Collects form data sent with POST method.
- $_GET → Collects form data sent with GET method.
- $_REQUEST → Collects data from both GET and POST (not recommended for security reasons).

Processing Example

```
<html>

    <body>

        <h1>Welcome <?php echo $_POST['name'];
?></h1>

        Your email address is <?php echo
$_POST['email']; ?>

    </body>

</html>
```

Output



```
Welcome John Doe

Your email address is john@example.com
```

Contd …

- PHP code that processes forms can be in the same file as the form or in a separate file.
- Same file example:

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
```

- This submits to the same file (common for validation).
- Security reminder: Never trust user input. Always validate and sanitize.

Working with HTTP Request (GET, POST, SERVER)

When a form is submitted, data is sent from the client (browser) to the server (PHP) using the HTTP protocol. PHP provides superglobal arrays to access this data.

GET Method

- The predefined $_GET variable use to collect values in a form with method=”get” information sent from a form with the GET method is visible to everyone (it will be displayed in the browser address bar) and has limits on the amount of information to send.
- $_GET Variable to collect form data (the name of the form field will automatically be the keys in the $_GET array)
- Data is sent as part of the URL query string.

Example:

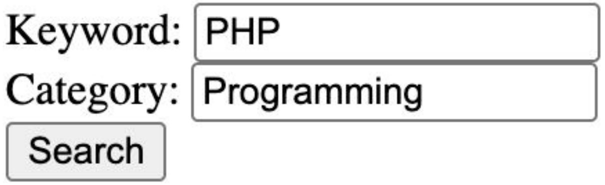
http://example.com/search.php?keyword=php&category=books

GET Method: Example

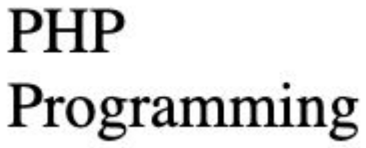
```
<form action="search.php" method="get">
  <input type="text" name="keyword">
  <input type="text" name="category">
  <button type="submit">Search</button>
</form>
```

```
<?php
echo $_GET['keyword'];
echo $_GET['category'];
?>
```

Output



Keyword: PHP
Category: Programming



```
PHP
Programming
```

POST Method

- The predefined $_POST Variable use to collect values in a form with method=”post” information sent from a form with the POST method is invisible to other and has no limits on the amount of information to send.
- Note: there is an 8MB max size for the POST method, by default (can be changed by setting the post_max_size in the php.ini file)
- $_POST Variable to collect form data (the name of the form field will automatically be the keys in the $_POST array)

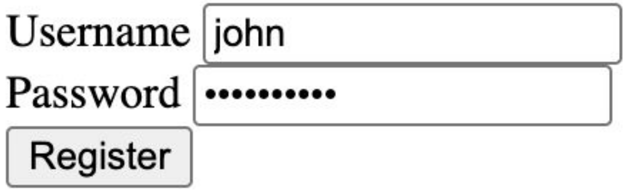
Example:

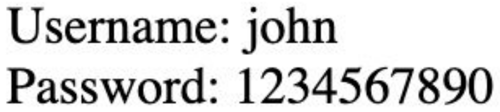
$_POST[“name”]

POST Method: Example

```
<form action="post/register.php"
method="post">
    <label>Username</label>
    <input type="text" name="username">
    <br>
    <label>Password</label>
    <input type="password" name="password">
    <br>
    <button type="submit">Register</button>
</form>
```

```
<?php

echo "Username: " . $_POST['username'];

echo "<br>";

echo "Password: " . $_POST['password'];

?>
```





```
Username: john
Password: 1234567890
```

Feature	GET	POST
Where data is sent	Appended to the URL as a query string.	Sent in the HTTP request body.
Visibility	Visible in the URL (anyone can see it).	Hidden from the URL.
Security	Less secure (not for passwords, sensitive data).	More secure (but still needs validation/sanitization).
Length Limit	Limited (around 2000 characters depending on browser/server).	Practically unlimited (good for large data and file uploads).
Bookmarkable	Yes, URLs can be saved and shared.	No, data is not part of the URL.
Use Cases	Search engines, filtering, navigation, passing non-sensitive parameters.	Login forms, registration, file uploads, contact forms.
Form Default	If `method` is not specified, form defaults to `GET`.	Must be explicitly declared using `method="post"`.

When to Use GET?

- Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
- GET may be used for sending non-sensitive data.
- Note: GET should NEVER be used for sending passwords or other sensitive information!

When to Use POST?

- Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request) and has no limits on the amount of information to send.
- Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.
- **Info:** Developers prefer POST for sending form data.

REQUEST Superglobal

- The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE. We will discuss $_COOKIE variable when we will explain about cookies.
- The PHP $_REQUEST variable can be used to get the result from form sent with both the GET and POST methods.

Example:

$_REQUEST[“name”]

Note: Not recommended for sensitive applications (can cause confusion, less secure).

SERVER Superglobal

- $_SERVER contains server and execution environment information.
- Commonly used variables:
  - $_SERVER['PHP_SELF'] → Current script filename.
  - $_SERVER['REQUEST_METHOD'] → GET or POST.
  - $_SERVER['HTTP_USER_AGENT'] → Browser info.
  - $_SERVER['REMOTE_ADDR'] → IP address of the client.
  - $_SERVER['SERVER_NAME'] → Host name of the server.

Example:

```
<?php
echo $_SERVER['REQUEST_METHOD'];   // POST or GET
echo $_SERVER['REMOTE_ADDR'];      // User's IP address
echo $_SERVER['HTTP_USER_AGENT'];  // Browser and OS info
```

Practical Example (Combining Concepts)

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST['username'];
    echo "Form submitted via POST. Username: " . $username;
} else {
    echo "Form not submitted yet or submitted via GET.";
}
?>
```

Dealing with Checkbox, Radio Button and List

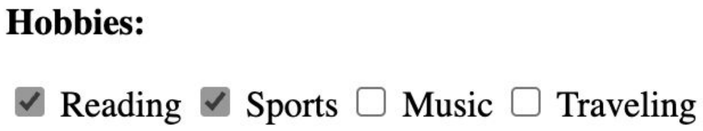
Checkbox

- A checkbox lets the user choose one or more options.
- If not checked → it won’t appear in $_POST.
- If checked → its value will be submitted.
- For multiple checkboxes, use array naming (name="hobbies[]").
- Examples:
  - Hobbies (sports, music, travel, etc.).
  - Newsletter subscriptions (email, SMS, WhatsApp).
  - Food toppings (cheese, mushrooms, olives).

Checkbox Example

```
<form method="post">
  <label><input type="checkbox"
name="hobbies[]" value="sports">
Sports</label>
  <label><input type="checkbox"
name="hobbies[]" value="music"> Music</label>
  <label><input type="checkbox"
name="hobbies[]" value="reading">
Reading</label>
  <button type="submit">Submit</button>
</form>
```

```
<?php
if (!empty($_POST['hobbies'])) {
    foreach ($_POST['hobbies'] as $hobby) {
        echo "Selected hobby: " .
htmlspecialchars($hobby) . "<br>";
    }
} else {
    echo "No hobbies selected.";
}
?>
```


Hobbies:

✓ Reading ✓ Sports ☐ Music ☐ Traveling

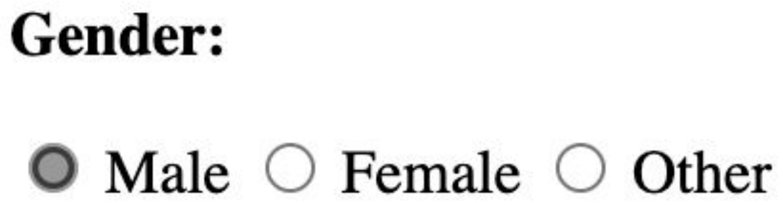
Radio Button

- A radio button allows only one option from a group.
- Must share the same name but have different values.
- If none is selected, that input won’t appear in $_POST.
- Examples:
    - Gender selection (male, female, other).
    - Payment method (credit card, PayPal, cash).
    - Delivery speed (standard, express, overnight).

Radio Button Example

```
<form method="post">
  <label><input type="radio" name="gender"
value="male"> Male</label>
  <label><input type="radio" name="gender"
value="female"> Female</label>
  <button type="submit">Submit</button>
</form>
```

```
<?php
if (isset($_POST['gender'])) {
    echo "Selected gender: " .
htmlspecialchars($_POST['gender']);
} else {
    echo "No gender selected.";
}
?>
```



Drop-down List (Select)

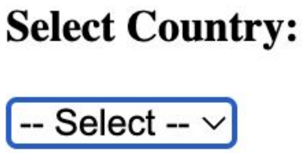
- A drop-down list (select box) lets users choose from a set of options.
- A drop-down list (<select>) lets users choose one option (default) or multiple options (if configured).
- Inside <select> → use <option> elements to define available choices.
- Each <option> has a value (submitted to the server) and optional label/text (shown to the user).
- There are two types of list:
    - Single-select
    - Multi-select

Single-select Example

- Allows the user to choose only one option.
- Common use cases: Country selection, gender, category filter, etc.

```
<form method="post">
  <select name="country">
    <option value="">--Select--</option>
    <option value="USA">USA</option>
    <option value="UK">UK</option>
    <option value="India">India</option>
  </select>
  <button type="submit">Submit</button>
</form>
```

```
<?php
if (!empty($_POST['country'])) {
    echo "Selected country: " .
htmlspecialchars($_POST['country']);
} else {
    echo "No country selected.";
}
?>
```



Multi-select Example

- Allows the user to select more than one option.
- Add the multiple attribute to <select>.
- The name must end with [] to send data as an array.

```
<form method="post">
  <select name="languages[]" multiple>
    <option value="PHP">PHP</option>
    <option value="Java">Java</option>
    <option value="Python">Python</option>
  </select>
  <button type="submit">Submit</button>
</form>
```

```
<?php
if (!empty($_POST['languages'])) {
    foreach ($_POST['languages'] as $lang) {
        echo "Selected language: " .
htmlspecialchars($lang) . "<br>";
    }
} else {
    echo "No language selected.";
}
?>
```

Preserving User Input

Why Preserve Input?

- Imagine a user fills a long registration form but makes a small mistake (e.g., missing email).
- Without preserved input, the form resets, and the user must re-enter everything.
- Preserving input means re-filling the form fields with previously submitted values so users only correct mistakes.
- This is essential for usability, form validation, and good UX design.

How it works?

When a form is submitted, the data is sent to the server (via POST or GET). If there are errors:

- The server sends the form back to the browser.
- The PHP script inserts the previously submitted data back into the form fields.
- The user sees their input intact and can correct only the errors.

How Forms Normally Behave

By default, when a form is submitted:

- The page reloads.
- All fields are cleared unless explicitly refilled.

Source Code:

```
<form action="" method="post">
   Name: <input type="text" name="name">
   <button type="submit">Submit</button>
</form>
```

Expected Outcome: After submission, the input disappears.

Preserving Text Input

Use PHP to re-populate the field with the submitted value.

Source Code:

```
<form action="" method="post">
  Name:
  <input type="text" name="name" value="<?php
      if (isset($_POST['name'])) echo htmlspecialchars($_POST['name']);
  ?>">
  <button type="submit">Submit</button>
</form>
```

Expected Outcome: If the user typed "John" and the form reloads, "John" remains inside the textbox.

Preserving Radio Buttons

Source Code:

```
Gender:
<input type="radio" name="gender" value="male"
  <?php if (isset($_POST['gender']) && $_POST['gender']=="male") echo
"checked"; ?>>
Male
<input type="radio" name="gender" value="female"
  <?php if (isset($_POST['gender']) && $_POST['gender']=="female") echo
"checked"; ?>>
Female
```

Expected Outcome:

Ensures the previously selected gender stays selected.

Preserving Checkboxes

Source Code:

```
Hobbies:
<input type="checkbox" name="hobbies[]" value="sports"
  <?php if (!empty($_POST['hobbies']) && in_array("sports", $_POST['hobbies']))
echo "checked"; ?>>
Sports
<input type="checkbox" name="hobbies[]" value="music"
  <?php if (!empty($_POST['hobbies']) && in_array("music", $_POST['hobbies']))
echo "checked"; ?>>
Music
```

Expected Outcome:

If the user checked "Sports" earlier, it stays checked after reload.

Preserving Drop-down (Select)

Source Code:

```
<select name="country">
  <option value="">--Select--</option>
  <option value="USA" <?php if(isset($_POST['country']) &&
$_POST['country']=="USA") echo "selected"; ?>>USA</option>
  <option value="UK" <?php if(isset($_POST['country']) &&
$_POST['country']=="UK") echo "selected"; ?>>UK</option>
  <option value="India" <?php if(isset($_POST['country']) &&
$_POST['country']=="India") echo "selected"; ?>>India</option>
</select>
```

Expected Outcome:

The user’s previously chosen country remains selected.

Validating and Sanitizing User’s Input

Why Validation is Needed

Validation is the process of checking if user input meets specific rules or requirements.

- Ensures required fields are filled
- To protect the website from illegal or invalid user input
- To protect the website from bad people
- Enhances user experience by guiding correct input

Types of Forms where Validation is Important

- Sign up Form
- Sign in Form
- Contact Form
- Request a Quote From
- Enquiry Form
- Any many more …

Types of Validation: Required Field Validation

Ensures the user does not leave an important field empty.

Source Code:

```
$name = isset($_POST['name']) ? $_POST['name'] : '';

if (empty($name)) {

    $errors['name'] = "Name is required.";

}
```

Email Validation

Check if the email has a valid format using filter_var():

Source Code:

```
$email = is_set($_POST['email']) ? $_POST['email'] : '';

if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {

    $errors['email'] = "Invalid email format.";

}
```

Number Validation

Ensure the input is numeric:

Source Code:

```
$age = is_set($_POST['age']) ? $_POST['age'] : '';

if (!is_numeric($age)) {

    $errors['age'] = "Age must be a number.";

}
```

String Length Validation

Check minimum or maximum characters for a field:

Source Code:

```
$password = $_POST['password'] ?? '';

if (strlen($password) < 6) {

    $errors['password'] = "Password must be at least 6 characters long.";

}
```

Pattern / Regex Validation

For complex rules, like phone numbers or usernames:

Source Code:

```
$phone = $_POST['phone'] ?? '';

if (!preg_match("/^[0-9]{10}$/", $phone)) {

    $errors['phone'] = "Phone must be 10 digits.";

}
```

Sanitization

Sanitization is the process of cleaning the input to remove harmful or unwanted characters.

Why Sanitization is Important?

- Protects against XSS attacks (Cross-Site Scripting).
- Prevents accidental injection of unwanted HTML or scripts.
- Ensures only clean data is stored in the database or displayed.

Sanitization Methods in PHP

Remove HTML Tags

```
$message = strip_tags($_POST['message'] ?? '');
```

Convert Special Characters

Prevents scripts from executing:

```
$name = htmlspecialchars($_POST['name'] ?? '');
```

- Converts < to &lt;, > to &gt;, etc.
- Essential when displaying user input in HTML.

Sanitization: Filter Input

PHP’s filter_var() can sanitize different types of data:

Source Code:

```
$email = filter_var($_POST['email'], FILTER_SANITIZE_EMAIL);

$url   = filter_var($_POST['website'], FILTER_SANITIZE_URL);

$int   = filter_var($_POST['age'], FILTER_SANITIZE_NUMBER_INT);
```

Combining Validation and Sanitization

Best practice: Always sanitize first, then validate.

Source Code:

```
// Sanitize input
$name = htmlspecialchars(trim($_POST['name'] ?? ''));
$email = filter_var($_POST['email'] ?? '', FILTER_SANITIZE_EMAIL);
// Validate input
$errors = [];
if (empty($name)) {
    $errors['name'] = "Name is required.";
}
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    $errors['email'] = "Invalid email format.";
}
```

Practical Tips

- Always assume user input is unsafe.
- Start with trimming whitespace: trim().
- Use filter_var() and htmlspecialchars() consistently.
- Check for required fields first before advanced validation.
- Store errors in an array and display them next to the relevant input.

File Upload

What is File Upload?

- File upload allows users to send files (images, documents, etc.) from their computer to the server via an HTML form.
- Common uses:
  - Profile picture upload.
  - Resume/CV upload.
  - Product images in e-commerce.
  - Assignment submissions in education systems.

What happens during file upload?

- User selects file from their computer
- Browser sends file to server via HTTP POST
- PHP temporarily stores file
- PHP validates and moves to permanent location

Server Configuration for File Upload

- Ensure that file uploads are enabled in your php.ini file. Look for the following directives and adjust them if necessary:

file_uploads = On

upload_max_filesize = 2M ; Or your desired maximum file size

post_max_size = 8M ; Should be greater than or equal to upload_max_filesize

max_file_uploads = 20 ; Maximum number of files that can be uploaded simultaneously

Form Requirements for File Upload

To handle file uploads, the HTML <form> must:

- Use method="post" (not GET).
- Use enctype="multipart/form-data" → required for sending file data.

```
<form action="upload.php" method="post"
enctype="multipart/form-data">

  <label for="file">Choose a file:</label>

  <input type="file" name="myfile" id="file">

  <button type="submit"
name="submit">Upload</button>

</form>
```

The $_FILES Superglobal

When a file is uploaded, PHP stores info in the $_FILES array.

Structure:

```
$_FILES['myfile']['name']      // Original filename
$_FILES['myfile']['type']      // MIME type (e.g., image/jpeg)
$_FILES['myfile']['size']      // Size in bytes
$_FILES['myfile']['tmp_name']  // Temporary location on server
$_FILES['myfile']['error']     // Error code (0 means no error)
```

Processing a File Upload (upload.php)

```
<?php
if (isset($_POST['submit'])) {
    $fileName = $_FILES['myfile']['name'];
    $fileTmp  = $_FILES['myfile']['tmp_name'];
    $fileSize = $_FILES['myfile']['size'];
    $fileError= $_FILES['myfile']['error'];

    if ($fileError === 0) {
        $destination = "uploads/" . basename($fileName);
        if (move_uploaded_file($fileTmp, $destination)) {
            echo "File uploaded successfully: " . htmlspecialchars($fileName);
        } else {
            echo "Error moving the file!";
        }
    } else {
        echo "File upload error code: " . $fileError;
    }
}
?>
```

Validating the File Upload

To make uploads safe and controlled, validate:

1. File Size Limit

```
if ($fileSize > 2 * 1024 * 1024) { // 2 MB limit
    echo "File too large!";
}
```

Validating the File Upload

2. Allowed File Types

```
$allowed = ['jpg','jpeg','png','pdf'];

$fileExt = strtolower(pathinfo($fileName,
PATHINFO_EXTENSION));



if (!in_array($fileExt, $allowed)) {

    echo "Invalid file type!";

}
```

3. MIME Type Check

```
if (mime_content_type($fileTmp) !==
'image/jpeg') {
    echo "Only JPEG images allowed!";
}
```

Or,

```
$allowedMimes = [
    'image/jpeg',
    'image/png',
    'image/gif',
    'application/pdf'
];

if (!in_array($fileType, $allowedMimes)) {
    echo "Invalid file type.";
}
```

File Upload Error Codes

Code	Constant	Meaning
0	UPLOAD_ERR_OK	No error, file uploaded successfully
1	UPLOAD_ERR_INI_SIZE	File exceeds upload_max_filesize in php.ini
2	UPLOAD_ERR_FORM_SIZE	File exceeds MAX_FILE_SIZE in HTML form
3	UPLOAD_ERR_PARTIAL	File only partially uploaded
4	UPLOAD_ERR_NO_FILE	No file was uploaded
6	UPLOAD_ERR_NO_TMP_DIR	Missing temporary folder
7	UPLOAD_ERR_CANT_WRITE	Failed to write file to disk

Handle Errors Properly

```
$fileError = $_FILES['myfile']['error'];

switch ($fileError) {
    case UPLOAD_ERR_OK:
        echo "File uploaded successfully!";
        break;
    case UPLOAD_ERR_NO_FILE:
        echo "No file was uploaded.";
        break;
    case UPLOAD_ERR_INI_SIZE:
    case UPLOAD_ERR_FORM_SIZE:
        echo "File is too large.";
        break;
    default:
        echo "Unknown error occurred.";
        break;
}
```

Send Email

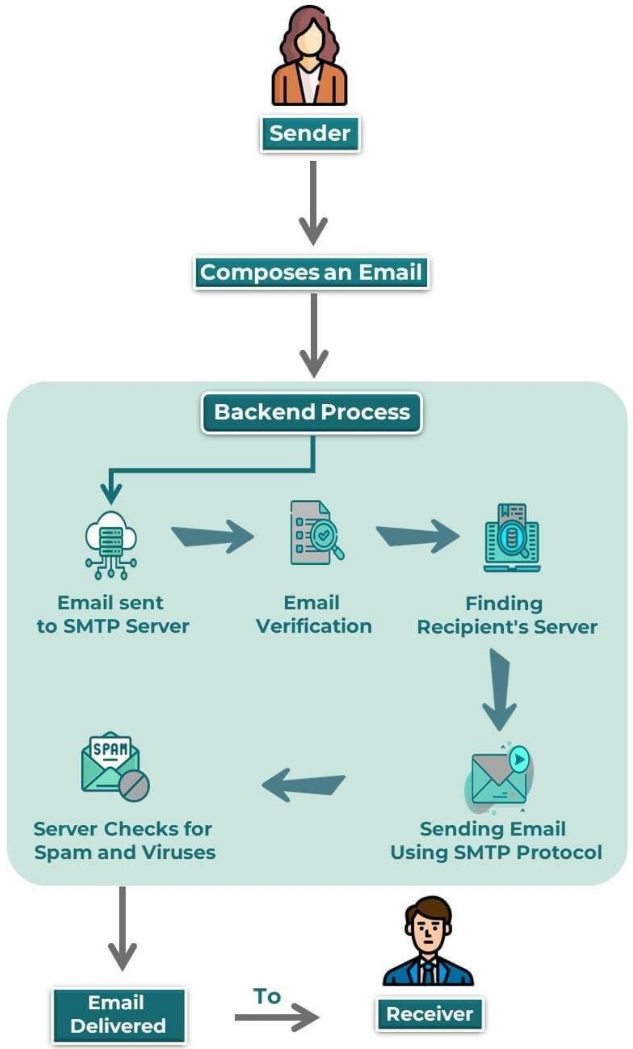
Why Send Emails?

Real-World Email Use Cases

- Contact form submissions.
- Sending OTPs or verification links.
- Order confirmations in e-commerce.
- Account verification emails (signup confirmation)
- Password reset links
- Newsletters and notifications
- Event tickets and bookings
- Reports and invoices
- Registration confirmations

How does Email work?

- PHP script creates email
- Sends to mail server (SMTP)
- Mail server delivers to recipient
- Recipient receives in inbox


Sender
Composes an Email
Backend Process
Email sent to SMTP Server
Email Verification
Finding Recipient's Server
Sending Email Using SMTP Protocol
Server Checks for Spam and Viruses
Email Delivered
To
Receiver

The mail() Function

PHP has a built-in function for sending emails:

mail(to, subject, message, headers, parameters);

Parameters:

- to → Recipient email address.
- subject → Subject of the email.
- message → Body text (plain text or HTML).
- headers (optional) → From, CC, BCC, content type, etc.
- parameters (optional) → Extra command-line flags (rarely used).

Basic Example

```php
<?php
$to = "student@example.com";
$subject = "Test Email";
$message = "Hello! This is a test email from PHP.";
$headers = "From: teacher@example.com";

if (mail($to, $subject, $message, $headers)) {
    echo "Email sent successfully!";
} else {
    echo "Failed to send email.";
}
?>
```

Sending HTML Emails

```
<?php
$to = "student@example.com";
$subject = "Welcome to the Course";
$message = "
<html>
<head><title>Welcome Email</title></head>
<body>
  <h2>Welcome to Internet Technology</h2>
  <p>Dear Student,</p>
  <p>Thanks for joining our course! </p>
</body>
</html>";
```

```
$headers  = "MIME-Version: 1.0" . "\r\n";

$headers .= "Content-type:text/html;charset=UTF-8"
. "\r\n";

$headers .= "From: teacher@example.com";



if (mail($to, $subject, $message, $headers)) {

    echo "HTML Email sent successfully!";

} else {

    echo "Failed to send email.";

}

?>
```

Contact Form Example: HTML Form:

```
<form method="post" action="sendmail.php">
  <input type="text" name="name" placeholder="Your Name"><br>
  <input type="email" name="email" placeholder="Your Email"><br>
  <textarea name="message" placeholder="Your Message"></textarea><br>
  <input type="submit" name="submit" value="Send Message">
</form>
```

Contact Form Example: PHP Script (sendmail.php)

```
<?php
if (isset($_POST['submit'])) {
    $name    = htmlspecialchars($_POST['name']);
    $email   = htmlspecialchars($_POST['email']);
    $message = htmlspecialchars($_POST['message']);

    $to = "teacher@example.com"; // Receiver
    $subject = "New Message from Contact Form";

    $body = "Name: $name\nEmail: $email\n\nMessage:\n$message";

    $headers = "From: $email";

    if (mail($to, $subject, $body, $headers)) {
        echo "Thank you $name, your message has been sent!";
    } else {
        echo "Sorry, failed to send your message.";
    }
}
?>
```

Limitations of mail()

- Works only if the server has a configured mail server (SMTP).
- On local XAMPP/WAMP → won’t work unless you configure SMTP.
- No built-in authentication (less secure).

Using PHPMailer (Better Approach)

For real-world apps, developers use PHPMailer or SwiftMailer because they:

- Support SMTP authentication.
- Allow attachments.
- Send HTML + plain text (multipart).
- Work with Gmail, Outlook, Yahoo, etc.

Example

```
use PHPMailer\PHPMailer\PHPMailer;
require 'vendor/autoload.php';

$mail = new PHPMailer(true);
$mail->isSMTP();
$mail->Host = 'smtp.gmail.com';
$mail->SMTPAuth = true;
$mail->Username = 'yourgmail@gmail.com';
$mail->Password = 'yourpassword'; // or App Password
$mail->SMTPSecure = 'tls';
$mail->Port = 587;

$mail->setFrom('yourgmail@gmail.com', 'Teacher');
$mail->addAddress('student@example.com');

$mail->Subject = 'Course Registration Successful';
$mail->Body    = 'Hello, your registration is confirmed!';

if($mail->send()) {
    echo "Email sent!";
} else {
    echo "Error: " . $mail->ErrorInfo;
}
```

Any Questions?