Ejercicio 1.- Crear un formulario que pida los parámetros para generar la contraseña:

Tendrá una serie de checks: que nos pida si quiere mayúsculas, minúsculas, números y símbolos. Puede marcar más de una opción y que longitud queremos que tenga la contraseña.

Finalmente mostraremos la contraseña generada aleatoriamente por medio de la librería password-generator

En el formulario deberá haber un botón que nos permita mostrar todas las contraseñas generadas.

Para desarrollar el ejercicio debemos cumplir los siguientes requisitos:

A) Utilizaremos la librería de https://github.com/hackzilla/password-generator

B) Crearemos dentro de nuestro proyecto la siguiente estructura

```
|-- /app
|  |-- /Clases
|  `-- /Interfaces
|
|-- /public
|  `
-- composer.json (Este archivo lo crearemos a continuación)
```

C) Utilizaremos composer y deberemos crear los siguientes ítems: Generar nombre del proyecto, descripción, el tipo que sea proyecto, el autor con el nombre y el correo.

Crearemos un espacio de nombres denominado “App” y que corresponda con el directorio “App”.

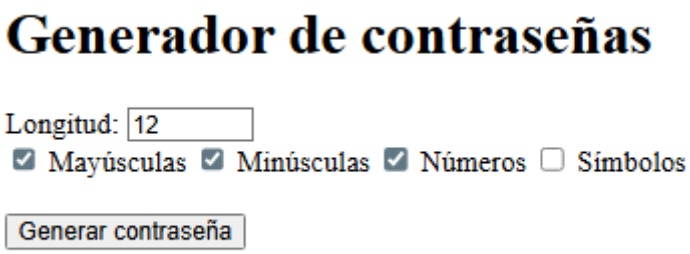
E) El formulario tiene de nombre “index.php” y debe figurar dentro del directorio public, el archivo que procesa el formulario debe llamarse procesa.php y también debe estar en public.

F) Dentro de app crearemos la carpeta Interfaces para contener InterfazGeneradorPassword que contiene una función generar

G) Dentro de app crearemos la carpeta Clases para contener la clase GeneradorPassword, que debe tener un método estático que genera la password con los criterios elegidos en el formulario y la clase AdaptadorGeneradorPassword que implementa la interfaz InterfazGeneradorPassword.

H) Documenta todo el proceso de creación del proyecto e instalación de composer y de la librería solicitada.

Index.php



Resultado de procesa.php

Ejercicio 2 .- Crear un formulario de contacto funcional que envíe su contenido a través de un correo electrónico, utilizando una arquitectura modular y las herramientas Composer y PHPMailer.

---

1. Estructura del Proyecto

Para empezar, organizaremos nuestro proyecto con la siguiente estructura de carpetas y archivos. Esto nos ayudará a mantener el código limpio y ordenado.

```
├── public/
│   ├── index.php          # Formulario de contacto (Frontend)
│   └── procesar.php       # Lógica de envío (Backend)
│
├── app/
│   ├── Clases/
│   │   ├── ServicioCorreo.php
│   │   └── ProveedorMailtrap.php
│   │
│   └── Interfaces/
│       └── InterfazProveedorCorreo.php
│
└── vendor/                # Carpeta autogenerada por Composer
```

---

2. Instalación de Dependencias con Composer

Utilizaremos **Composer** para gestionar las librerías externas de nuestro proyecto. La dependencia principal será **PHPMailer**, una de las bibliotecas más populares para el envío de correos en PHP.

- Abre una terminal en la raíz de tu proyecto y ejecuta:
  `composer require phpmailer/phpmailer`
- A lo largo del proyecto, haremos uso de **espacios de nombres (namespaces)** para organizar nuestras clases e interfaces de forma eficiente.

Consulta: https://github.com/PHPMailer/PHPMailer Encontraras ejemplos de uso, uno está incluido al final de esta explicación.

---

3. Creación de la Interfaz de Proveedor de Correo

Para que nuestra aplicación no dependa de un único servicio de correo (como Mailtrap, Gmail, etc.), crearemos una interfaz. Esto nos permitirá cambiar de proveedor en el futuro sin modificar la lógica principal.

- **Archivo:** `app/Interfaces/InterfazProveedorCorreo.php`
- **Contenido:**
  - Define un método `enviarCorreo()` que debe ser implementado por cualquier clase de proveedor.
  - **Parámetros:**
    - `string $paraQuien`: El destinatario del correo.
    - `string $asunto`: El asunto del mensaje.
    - `string $cuerpoMensaje`: El contenido del correo.
  - **Retorno:**
    - `bool`: `true` si el correo se envió correctamente, `false` en caso de error.

---

4. Desarrollo del Servicio de Envío de Correo

Este servicio actuará como el orquestador, utilizando la interfaz para enviar el correo sin conocer los detalles del proveedor específico.

- **Archivo:** `app/Clases/ServicioCorreo.php`
- **Contenido:**
  - **Constructor:** Recibirá un objeto de una clase que implemente `InterfazProveedorCorreo` y lo guardará como una propiedad privada de solo lectura.
  - **Método `enviarCorreo()`:** Llamará al método `enviarCorreo()` del proveedor que se le haya pasado en el constructor.

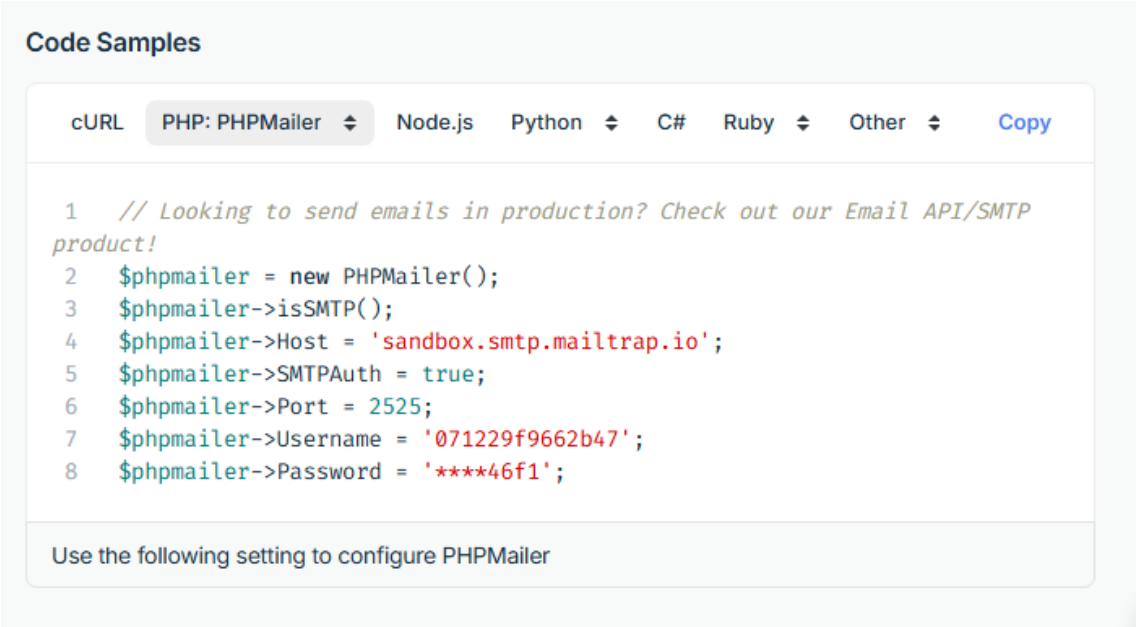
```php
<?php
declare(strict_types=1);

namespace App\Clases;

use App\Interfaces\InterfazProveedorCorreo;

/**
 * Servicio que orquesta el envío de correos usando un proveedor que implemente la interfaz.
 */
class ServicioCorreo
{
    private InterfazProveedorCorreo $proveedor;

    public function __construct(InterfazProveedorCorreo $proveedor)
    {
        $this->proveedor = $proveedor;
    }

    /**
     * Delegar el envío al proveedor.
     */
    public function enviarCorreo(string $paraQuien, string $asunto, string $cuerpoMensaje): bool
    {
        return $this->proveedor->enviarCorreo($paraQuien, $asunto, $cuerpoMensaje);
    }
}
```

---

5. Implementación del Proveedor: Mailtrap

Usaremos **Mailtrap**, un servicio que captura los correos salientes en un "buzón falso", evitando enviar correos reales a los destinatarios.

1. **Crear una cuenta en Mailtrap:**
   - Visita https://mailtrap.io/ y regístrate (puedes usar tu correo de Educantabria).
   - Dentro de "Email Testing" o “Sandbox”, busca tu bandeja de entrada y tus credenciales SMTP, selecciona PHPMailer



2. **Crear la clase del proveedor:**
   - **Archivo:** `src/Clases/ProveedorMailtrap.php`
   - **Funcionalidad:**
     - Debe implementar la `InterfazProveedorCorreo`.
     - Utiliza los **espacios de nombres** de PHPMailer: `PHPMailer\PHPMailer\PHPMailer`, `PHPMailer\PHPMailer\SMTP`, `PHPMailer\PHPMailer\Exception`.
     - Desarrolla el método `enviarCorreo()`, configurando PHPMailer con las credenciales SMTP de tu cuenta de Mailtrap.

---

6. Creación del Formulario (`index.php`)

Este será el punto de entrada para el usuario.

- **Archivo:** `public/index.php`
- **Campos requeridos:**
  - Nombre (`<input type="text">`)
  - Correo electrónico (`<input type="text">`) - *Se validará que sea un email válido en el backend.*
  - Mensaje (`<textarea>`)
- **Gestión de mensajes:** El formulario deberá mostrar notificaciones al usuario según los parámetros que reciba por `GET`:
  - `?error=1`: "Por favor, rellena todos los campos."
  - `?error=2`: "Por favor, introduce un email válido."
  - `?error=3`: "Ha ocurrido un error al enviar el email."
  - `?success=1`: "El email se ha enviado correctamente."

```
<!-- Mensajes según GET -->
   <?php if (isset($_GET['error']) && $_GET['error'] === '1'): ?>
       <p style="color:#b00">Por favor, rellena todos los campos.</p>
```

---

7. Lógica del Backend (`procesar.php`)

Este archivo recibirá los datos del formulario, los validará y coordinará el envío del correo.

- **Archivo:** `public/procesar.php`
- **Pasos a seguir:**
  1. **Autoload de Composer:** Incluye el `autoload.php` de Composer para cargar automáticamente las clases (`ServicioCorreo.php`, `ProveedorMailtrap.php`, etc.) siguiendo el estándar PSR-4.
  2. **Validación de campos:**
     - Comprueba que los tres campos (nombre, correo, mensaje) no estén vacíos. Si alguno lo está, redirige a `index.php?error=1`.

```
// Campos obligatorios
if ($nombre === '' || $email === '' || $mensaje === '') {
    header('Location: index.php?error=1');
    exit;
}
```

- Valida que el formato del correo sea correcto usando `filter_var($email, FILTER_VALIDATE_EMAIL)`. Si no es válido, redirige a `index.php?error=2`.

3. **Instanciación de Clases:**
   - Crea una instancia de `ProveedorMailtrap`.
   - Crea una instancia de `ServicioCorreo`, pasándole el objeto `ProveedorMailtrap` en el constructor.
4. **Envío del Correo:**
   - Llama al método `enviarCorreo()` del servicio, pasándole el destinatario, el asunto y el cuerpo del mensaje.
5. **Redirección Final:**
   - Si el correo se envió con éxito, redirige a `index.php?success=1`.
   - Si hubo un fallo en el envío, redirige a `index.php?error=3`.

Nota: Se adjunta a la actividad un ejemplo `enviar_correo.php` que puede servir como guía para la configuración específica de PHPMailer con Mailtrap.

```php
<?php
//Import PHPMailer classes into the global namespace
//These must be at the top of your script, not inside a function
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\SMTP;
use PHPMailer\PHPMailer\Exception;

//Load Composer's autoloader (created by composer, not included with
PHPMailer)
require 'vendor/autoload.php';

//Create an instance; passing `true` enables exceptions
$mail = new PHPMailer(true);

try {
    //Server settings
    $mail->SMTPDebug = SMTP::DEBUG_SERVER; //Enable verbose debug output
    $mail->isSMTP();       //Send using SMTP
    $mail->Host       = 'smtp.example.com';//Set the SMTP server to send
through
    $mail->SMTPAuth   = true;          //Enable SMTP authentication
    $mail->Username   = 'user@example.com';       //SMTP username
    $mail->Password   = 'secret';           //SMTP password
```

```
    $mail->SMTPSecure = PHPMailer::ENCRYPTION_SMTPS; //Enable implicit
TLS encryption
    $mail->Port       = 465; //TCP port to connect to; use 587 if you
have set `SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS`

    //Recipients
    $mail->setFrom('from@example.com', 'Mailer');
    $mail->addAddress('joe@example.net', 'Joe User'); //Add a recipient
    $mail->addAddress('ellen@example.com');    //Name is optional
    $mail->addReplyTo('info@example.com', 'Information');
    $mail->addCC('cc@example.com');
    $mail->addBCC('bcc@example.com');

    //Attachments
    $mail->addAttachment('/var/tmp/file.tar.gz');    //Add attachments
    $mail->addAttachment('/tmp/image.jpg', 'new.jpg');   //Optional name

    //Content
    $mail->isHTML(true);             //Set email format to HTML
    $mail->Subject = 'Here is the subject';
    $mail->Body    = 'This is the HTML message body <b>in bold!</b>';
    $mail->AltBody = 'This is the body in plain text for non-HTML mail
clients';

    $mail->send();
    echo 'Message has been sent';
} catch (Exception $e) {
    echo "Message could not be sent. Mailer Error: {$mail->ErrorInfo}";
}
```