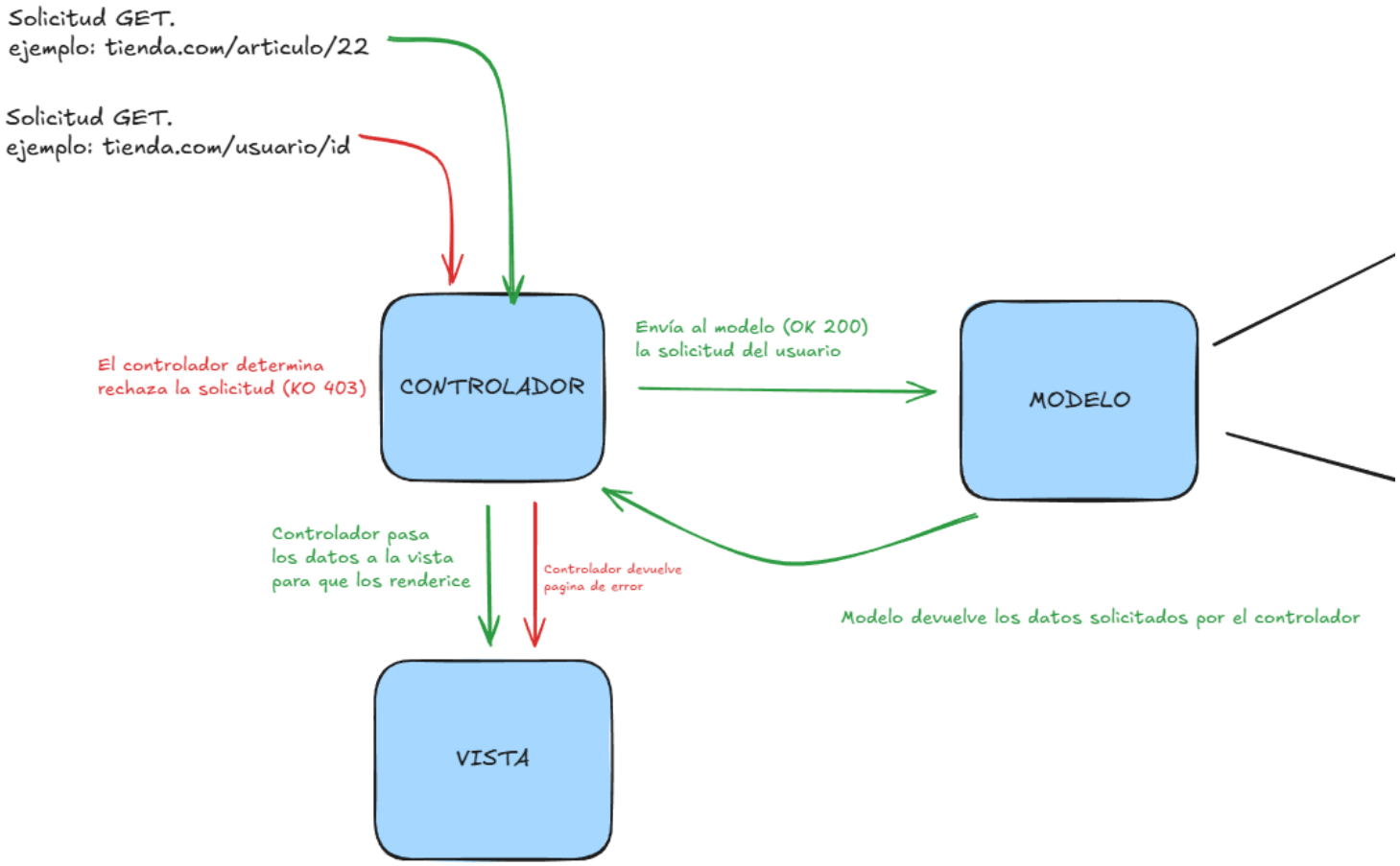
MVC - Model View Controller

Roles de cada componente

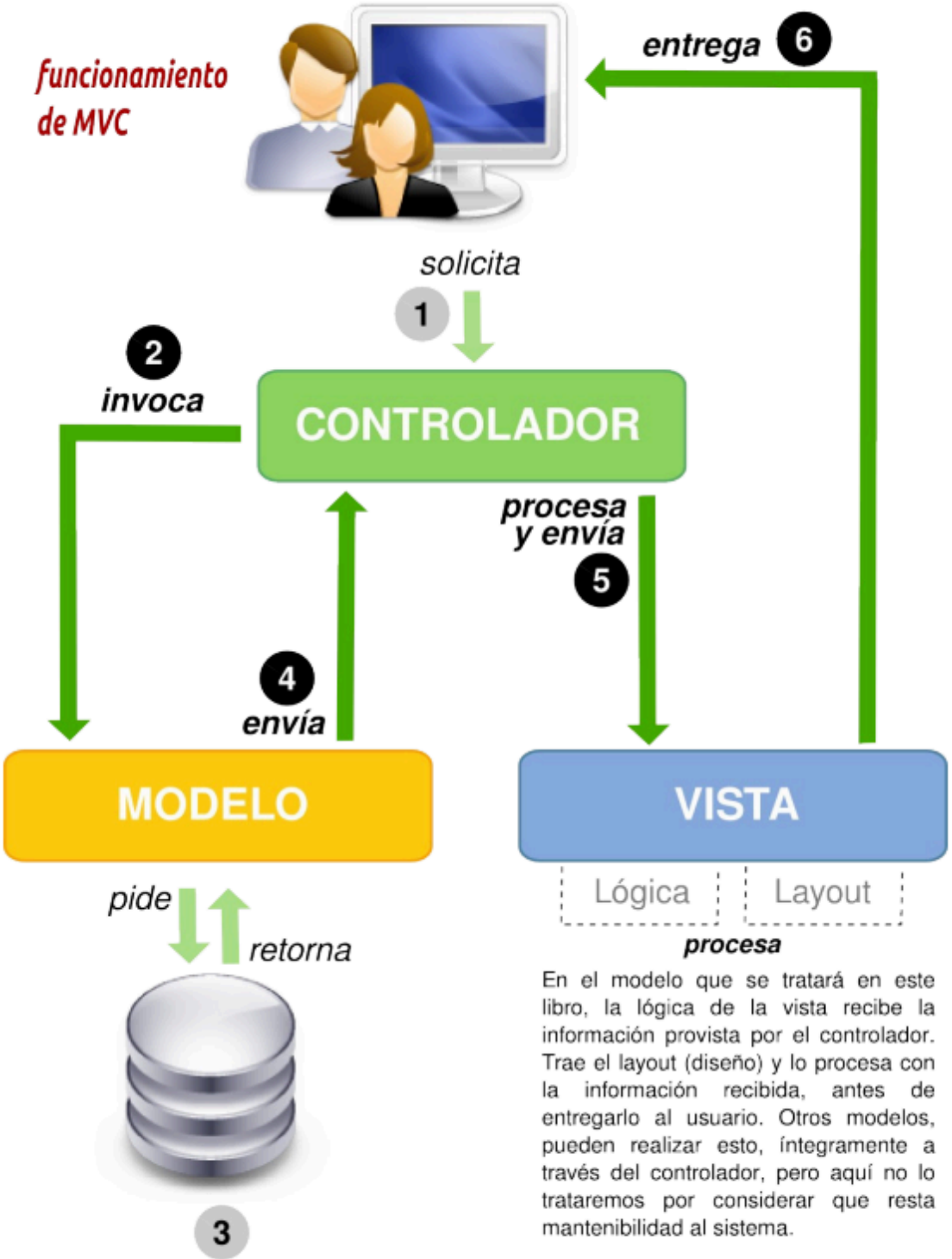
- **Modelo: Encapsula la lógica de negocio y el acceso a datos. Recupera, valida y transforma datos del almacenamiento, pero no decide cómo se presentan.**
- **Controlador: Orquesta el flujo. Recibe la petición, invoca al Modelo, recibe los datos y los adapta para la Vista (formateo, agregaciones, selección de campos, manejo de estados y errores).**
- **Vista: Presenta la información. Renderiza plantillas con los datos que le proporciona el Controlador, sin lógica de negocio.**

Secuencia detallada del flujo

1. **Llegada de la petición:**
    - **Evento: El usuario realiza una acción (ej. GET /productos/42).**
    - **Responsable: Controlador recibe la petición con parámetros, cabeceras, sesión, etc.**
2. **Invocación al Modelo:**
    - **Acción: El Controlador solicita al Modelo los datos necesarios (consultas, validaciones, reglas de dominio).**
3. **Recepción y procesamiento en el Controlador:**
    - **Acción: El Controlador recibe el resultado del Modelo.**
    - **Procesos típicos:**
        - **Mapeo: Convierte entidades a DTOs o estructuras aptas para la Vista.**
        - **Formateo: Fechas, monedas, i18n, paginación.**
        - **Composición: Une datos de varias fuentes o aplica filtros.**
        - **Gestión de errores: Decide qué Vista (o estado HTTP) renderizar en caso de fallos.**
4. **Envío a la Vista:**
    - **Acción: El Controlador selecciona la Vista (plantilla o componente) y le pasa el modelo de presentación (datos ya listos para mostrar).**
    - **Resultado: La Vista renderiza HTML/JSON/XML según el caso.**



Página web renderizada que vería el usuario



En el modelo que se tratará en este libro, la lógica de la vista recibe la información provista por el controlador. Trae el layout (diseño) y lo procesa con la información recibida, antes de entregarlo al usuario. Otros modelos, pueden realizar esto, íntegramente a través del controlador, pero aquí no lo trataremos por considerar que resta mantenibilidad al sistema.

Funcionamiento del patrón modelo-vista-controlador

```
[Navegador] --> http://localhost:8010/ejemplo
      |
      v
[Docker Compose] --> puerto 8010 --> contenedor PHP+Apache
      |
      v
[Apache] recibe /ejemplo
      |
redirige a index.php
      |
      v
[PHP] ejecuta index.php
      |
      v
index.php
  |
  |-- require vendor/autoload.php
  |-- Dotenv::load()  -> variables de entorno disponibles
  |-- new RouteCollection() -> mapa de rutas
  |-- new Request() -> route + params desde REQUEST_URI
  |-- new Dispatcher(route, request)
          |
          |-- request->getRoute()
          |-- route->getRoutes()[route] => controller/action
          |-- new Controller
          |-- request->getParams()
          |-- controller->action(params...)
          |-- render/responder
```

COMPOSER:

```
"name": "daw-dwes/ap5-1-mvc-doctrine-crud",
"authors": [
  {
    "name": "lluís alandete",
    "email": "lalandete@florida-uni.es"
  }
],
"autoload": {
  "psr-4": {
    "CAMBIAME\\": "src/"
  }
},
"require": {
  "vlucas/phpdotenv": "^5.6",
  "doctrine/orm": "^3",
  "doctrine/dbal": "^4",
  "symfony/cache": "^7"
}
}
```

—---

1️⃣ ¿Qué es Doctrine?

Doctrine es una **librería de PHP** muy popular, usada normalmente con Symfony, que permite trabajar con bases de datos de manera más **orientada a objetos**, evitando escribir demasiado SQL manual.

2️⃣ ¿Qué significa ORM?

ORM significa **Object-Relational Mapping** (Mapeo Objeto-Relacional).
Básicamente, es **una técnica de programación que permite que las tablas de la base de datos se "mapen" a objetos en tu código**. Esto tiene varias ventajas:

- No tienes que escribir SQL todo el tiempo.
- Puedes trabajar con objetos de PHP como si fueran entidades reales, y Doctrine se encarga de convertir esos objetos en registros en la base de datos y viceversa.
- Mantiene tu código más limpio y orientado a objetos.

Ejemplo simple:

Supongamos que tienes una tabla `usuarios` con columnas `id`, `nombre` y `email`. Con Doctrine ORM, podrías tener una clase `Usuario` así:

```
class Usuario
{
    private $id;
    private $nombre;
    private $email;

    // getters y setters
}
```

Doctrine se encarga de mapear esta clase a la tabla `usuarios` y sincronizar los cambios en objetos con la base de datos.

3️⃣ Analizando las opciones

1. **Un componente de Symfony para la gestión de la base de datos.**
   ❌ Incorrecto. Doctrine **no es parte del core de Symfony**, aunque se integra fácilmente. Es una librería independiente.
2. **Una técnica de programación que permite mapear tablas de bases de datos a objetos de una aplicación.**
   ✅ Correcto. Esto es exactamente lo que hace un ORM.
3. **Un conjunto de métodos para realizar consultas SQL nativas.**
   ❌ Incorrecto. Doctrine tiene soporte para SQL nativo, pero eso **no define lo que es un ORM**.
4. **La capa de abstracción para la base de datos.**
   ⚠️ Parcialmente cierto. Doctrine incluye una **capa de abstracción**, pero el término ORM se refiere específicamente al **mapeo objeto-relacional**, no solo a la abstracción.

✅ Conclusión

La respuesta correcta es:

“Una técnica de programación que permite mapear tablas de bases de datos a objetos de una aplicación.”

Ejemplo: *OneToOne* ***unidireccional*** *de libros a usuarios.*

Clase referenciada

```php
#[Entity]
#[Table(name: "usuarios")]
class Usuario
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 64)]
    private string $nombre;
```

Clase que tiene la relación

```php
#[Entity]
#[Table(name: "libros")]
class Libro
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 255)]
    private string $titulo;

    #[OneToOne(targetEntity: Usuario::class)]
    #[JoinColumn(name: "usuario_id", referencedColumnName: "id", nullable: false)]
    private Usuario $usuario;
```

Ejemplo: *OneToOne* ***bidireccional****. Libros es el principal.*

Clase referenciada

```php
#[Entity]
#[Table(name: "usuarios")]
class Usuario
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 64)]
    private string $nombre;

    #[OneToOne(mappedBy: "usuario", targetEntity: Libro::class)]
    private ?Libro $libro = null;
```

Clase que tiene la relación

```php
#[Entity]
#[Table(name: "libros")]
class Libro
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 255)]
    private string $titulo;

    #[OneToOne(inversedBy: "libro", targetEntity: Usuario::class)]
    #[JoinColumn(name: "usuario_id", referencedColumnName: "id", nullable: false)]
    private ?Usuario $usuario = null;
```

En la BD esta es la tabla que tiene la FK

Ejemplo: *OneToMany/ManyToOne* ***unidireccional****. Un usuario puede tener muchos libros*

Clase referenciada

```php
#[Entity]
#[Table(name: "usuarios")]
class Usuario
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 64)]
    private string $nombre;
```

Clase que tiene la relación

```php
#[Entity]
#[Table(name: "libros")]
class Libro
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 255)]
    private string $titulo;

    #[ManyToOne(targetEntity: Usuario::class, inversedBy: "libros")]
    #[JoinColumn(name: "usuario_id", referencedColumnName: "id", nullable: false)]
    private ?Usuario $usuario = null;
```

6. Asociaciones OneToMany (ManyToOne)

Ejemplo: *OneToMany/ManyToOne* ***bidireccional****. Un usuario puede tener muchos libros*

Clase referenciada

```php
#[Entity]
#[Table(name: "usuarios")]
class Usuario
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 64)]
    private string $nombre;

    #[OneToMany(mappedBy: "usuario", targetEntity: Libro::class, cascade: ["persist", "remove"])]
    private Collection $libros;
```

Clase que tiene la relación

```php
#[Entity]
#[Table(name: "libros")]
class Libro
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 255)]
    private string $titulo;

    #[ManyToOne(targetEntity: Usuario::class, inversedBy: "libros")]
    #[JoinColumn(name: "usuario_id", referencedColumnName: "id", nullable: false, onDelete: "CASCADE")]
    private ?Usuario $usuario = null;
```

7. Asociaciones ManyToMany

Ejemplo: *ManyToMany* ***unidireccional***

```php
#[Entity]
#[Table(name: "usuarios")]
class Usuario
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 64)]
    private string $nombre;

    #[ManyToMany(targetEntity: Libro::class, cascade: ["persist", "remove"])]
    #[JoinTable(
        name: "usuarios_libros",
        joinColumns: [new JoinColumn(name: "usuario_id", referencedColumnName: "id")],
        inverseJoinColumns: [new JoinColumn(name: "libro_id", referencedColumnName: "id")]
    )]
    private Collection $libros;
```

```php
#[Entity]
#[Table(name: "libros")]
class Libro
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 255)]
    private string $titulo;
}
```

7. Asociaciones ManyToMany

Ejemplo:** ManyToMany **bidireccional

```php
#[Entity]
#[Table(name: "usuarios")]
class Usuario
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 64)]
    private string $nombre;

    #[ManyToMany(targetEntity: Libro::class, inversedBy: "usuarios")]
    #[JoinTable(
        name: "usuarios_libros",
        joinColumns: [new JoinColumn(name: "usuario_id", referencedColumnName: "id")],
        inverseJoinColumns: [new JoinColumn(name: "libro_id", referencedColumnName: "id")]
    )]
    private Collection $libros;
```

```php
#[Entity]
#[Table(name: "libros")]
class Libro
{
    #[Id]
    #[GeneratedValue]
    #[Column(type: "integer")]
    private int $id;

    #[Column(type: "string", length: 255)]
    private string $titulo;

    #[ManyToMany(targetEntity: Usuario::class, mappedBy: "libros")]
    private Collection $usuarios;
```

JoinTable() podría estar en el otro lado.

Ejemplo de CRUD con el controlador de usuario:

```php
<?php

namespace AP52\Controllers;

use AP52\Core\EntityManager;
use AP52\Entity\User;
use AP52\Repository\UserRepository;
use AP52\Views\ListUsersView;
use AP52\Views\FormUserView;
use AP52\Views\DeleteUserView;

class UserController
{
    private EntityManager $entityManager;
    private UserRepository $repository;

    public function __construct()
    {
        $this->entityManager = new EntityManager();
        $this->repository =
$this->entityManager->getEntityManager()->getRepository(User::clas
s);
    }
```

```php
    /**
     * Lista todos los usuarios
     */
    public function list(): void
    {
        $users = $this->repository->findAll();
        $view = new ListUsersView();
        $view->render($users);
    }

    /**
     * Gestiona las operaciones CRUD según los parámetros recibidos
     */
    public function crud(...$params): void
    {
        $action = $params[0] ?? null;
        $id = $params[1] ?? null;

        switch ($action) {
            case 'create':
                $this->create();
                break;
            case 'update':
                $this->update($id);
                break;
            case 'delete':
                $this->delete($id);
                break;
            default:
                $this->noRuta();
        }
    }

    /**
     * Crea un nuevo usuario
     */
    private function create(): void
    {
        if (isset($_POST['submit'])) {
            // Validar campos requeridos
            if (!isset($_POST['username'], $_POST['name'],
$_POST['first_subname'],
                       $_POST['country'], $_POST['email']) ||
                empty($_POST['username']) || empty($_POST['name'])
||
                empty($_POST['first_subname']) ||
empty($_POST['country']) ||
                empty($_POST['email'])) {
```

```
                $this->noRuta();
                return;
            }

            $user = new User();
            $user->setUsername($_POST['username']); // Cambiamos el
nombre ProductName
            $user->setName($_POST['name']);
            $user->setFirstSubname($_POST['first_subname']);
            $user->setSecondSubname($_POST['second_subname'] ??
null);
            $user->setAddress($_POST['address'] ?? null);
            $user->setTelephone($_POST['telephone'] ?? null);
            $user->setCity($_POST['city'] ?? null);
            $user->setCountry($_POST['country']);
            $user->setObservation($_POST['observation'] ?? null);
            $user->setEmail($_POST['email']);

            $em = $this->entityManager->getEntityManager();
            $em->persist($user);
            $em->flush();

            $this->list();
        } else {
            $view = new FormUserView();
            $view->render(false, null);
        }
    }

    /**
     * Actualiza un usuario existente
     */
    private function update(?string $id): void
    {
        $userId = intval($id);
        $user = $this->repository->find($userId);

        if (!$user) {
            $this->noRuta();
            return;
        }

        if (isset($_POST['submit'])) {
            if (!isset($_POST['username'], $_POST['name'],
$_POST['first_subname'],
                       $_POST['country'], $_POST['email']) ||
                empty($_POST['username']) || empty($_POST['name'])
||
```

```
                empty($_POST['first_subname']) ||
empty($_POST['country']) ||
                empty($_POST['email'])) {
                $this->noRuta();
                return;
            }

            $user->setUsername($_POST['username']);
            $user->setName($_POST['name']);
            $user->setFirstSubname($_POST['first_subname']);
            $user->setSecondSubname($_POST['second_subname'] ??
null);
            $user->setAddress($_POST['address'] ?? null);
            $user->setTelephone($_POST['telephone'] ?? null);
            $user->setCity($_POST['city'] ?? null);
            $user->setCountry($_POST['country']);
            $user->setObservation($_POST['observation'] ?? null);
            $user->setEmail($_POST['email']);

            $em = $this->entityManager->getEntityManager();
            $em->flush();

            $this->list();
        } else {
            $view = new FormUserView();
            $view->render(true, $user);
        }
    }

    /**
     * Elimina un usuario
     */
    private function delete(?string $id): void
    {
        $userId = intval($id);
        $user = $this->repository->find($userId);

        if (!$user) {
            $this->noRuta();
            return;
        }

        if (isset($_POST['confirm'])) {
            try {
                $em = $this->entityManager->getEntityManager();
                $em->remove($user);
                $em->flush();
```

```
                $this->list();
            } catch (\Exception $e) {
                $view = new DeleteUserView();
                $error = "No se puede eliminar el usuario porque
tiene conexiones asociadas.";
                $view->render($user, $error);
            }
        } else {
            $view = new DeleteUserView();
            $view->render($user);
        }
    }

    private function noRuta()
    {
        (new MainController)->noRuta();
    }
}
```