1. Objetivo del Proyecto

Gestión Integral de una tienda Online

- Vas a desarrollar una **aplicación web modular en PHP puro** (sin frameworks, que se usarán en el segundo cuatrimestre).
- La arquitectura será profesional y moderna: modelos para los datos, servicios para la lógica, archivos de configuración, vistas/controladores para la presentación y pruebas automatizadas.
- **Objetivo:** crear una **gestión completa de una tienda online** con gestión de productos, categorías y usuarios.
- **CRUD de Productos:**
  Implementa las operaciones fundamentales sobre los productos:
  - **Crear:** Añadir nuevos productos al catálogo.
  - **Leer:** Consultar el listado de productos y detalles individuales.
  - **Actualizar:** Modificar la información existente de los productos.
  - **Eliminar:** Borrar productos del sistema.

  Todas estas operaciones se realizarán en PHP y estarán protegidas por seguridad: solo usuarios autenticados y con el rol adecuado (por ejemplo, administrador) podrán modificar o eliminar productos.
- **Seguridad:**
  - Los usuarios deben autenticarse (login) para poder acceder a la web.
  - Sólo los usuarios con permisos adecuados (según su rol: usuario o administrador) podrán crear, modificar o eliminar productos.
  - Habrá operaciones que serán exclusivas del "admin".

2. Configuración Estructura Proyecto

2.1. Estructura Carpetas y Archivos

Estructura de carpetas recomendadas para tu proyecto

1. public/
   1.1 index.php

2. app(o src)/
   2.1 config/
      2.1.1 Config.php
   2.2 services/
      2.2.1 SessionService.php
      2.2.2 UsersService.php
      2.2.3 ProductosService.php
      2.2.4 CategoriasService.php
   2.3 uploads/ (donde se guardan las imágenes de productos)
   2.4 header.php
   2.5 footer.php
   2.6 login.php
   2.7 logout.php
   2.8 create.php
   2.9 update.php
   2.10 update-image.php
   2.11 update_image_file.php
   2.12 delete.php
   2.13 details.php
3.database
4.vendor (se instala automaticamente al lanzar composer install)

2.2. Contenido de los Ficheros .Nivel Raiz

Carpeta / Archivo	Contenido o subcarpetas / archivos internos
database/	**init.sql** (Este sería un ejemplo de init.sql , tu tendrás que adaptarlo a tu proyecto)

```
SELECT 'CREATE DATABASE nombre_de_la_base_de_datos'
    WHERE NOT EXISTS (SELECT FROM pg_database WHERE datname = 'tienda'

DROP TABLE IF EXISTS "productos";
DROP SEQUENCE IF EXISTS productos_id_seq;
DROP TABLE IF EXISTS "user_roles";
DROP TABLE IF EXISTS "usuarios";
DROP SEQUENCE IF EXISTS usuarios_id_seq;
DROP TABLE IF EXISTS "categorias";

-- Cuiddado con las secuencias, si se borran se pierde el autoincremen
CREATE SEQUENCE productos_id_seq INCREMENT 1 MINVALUE 1 MAXVALUE 92233

CREATE TABLE "public"."productos"
(
    "is_deleted"   boolean          DEFAULT false,
    "precio"       double precision DEFAULT '0.0',
    "stock"        integer          DEFAULT '0',
    "created_at"   timestamp        DEFAULT CURRENT_TIMESTAMP
    "id"           bigint           DEFAULT nextval('productos_id_seq'
    "updated_at"   timestamp        DEFAULT CURRENT_TIMESTAMP
    "categoria_id" uuid,
    "uuid"         uuid
    "descripcion"  character varying(255),
    "imagen"       text             DEFAULT 'https://via.placeholder.c
    "marca"        character varying(255),
    "modelo"       character varying(255),
    CONSTRAINT "productos_pkey" PRIMARY KEY ("id"),
    CONSTRAINT "productos_uuid_key" UNIQUE ("uuid")
) WITH (oids = false);

INSERT INTO "productos" ("is_deleted", "precio", "stock", "created_at"
                         "descripcion", "imagen", "marca", "modelo")
VALUES ('f', 10.99, 5, '2023-11-02 11:43:24.722473', 1, '2023-11-02 11
        'd69cf3db-b77d-4181-b3cd-5ca8107fb6a9', '19135792-b778-441f-87
        'https://via.placeholder.com/150', 'Nike', 'Modelo1'),
       ('f', 19.99, 10, '2023-11-02 11:43:24.722473', 2, '2023-11-02 1
        '6dbcbf5e-8e1c-47cc-8578-7b0a33ebc154', '662ed342-de99-45c6-84
        'https://via.placeholder.com/150', 'Adidas', 'Modelo2'),
       ('f', 15.99, 2, '2023-11-02 11:43:24.722473', 3, '2023-11-02 11
        'd69cf3db-b77d-4181-b3cd-5ca8107fb6a9', 'b79182ad-91c3-46e8-90
        'https://via.placeholder.com/150', 'Nike', 'Modelo3'),
       ('f', 25.99, 8, '2023-11-02 11:43:24.722473', 4, '2023-11-02 11
```

Carpeta / Archivo	Contenido o subcarpetas / archivos internos

```
         '6dbcbf5e-8e1c-47cc-8578-7b0a33ebc154', '4fa72b3f-dca2-4fd8-b8
         'https://via.placeholder.com/150', 'Nike', 'Modelo4'),
        ('f', 12.99, 3, '2023-11-02 11:43:24.722473', 5, '2023-11-02 11
         '6dbcbf5e-8e1c-47cc-8578-7b0a33ebc154', '1e2584d8-db52-45da-b2
         'https://via.placeholder.com/150', 'Adidas', 'Modelo5');


CREATE TABLE "public"."user_roles"
(
    "user_id" bigint NOT NULL,
    "roles"   character varying(255)
) WITH (oids = false);

INSERT INTO "user_roles" ("user_id", "roles")
VALUES (1, 'USER'),
       (1, 'ADMIN'),
       (2, 'USER'),
       (2, 'USER'),
       (3, 'USER');

CREATE SEQUENCE usuarios_id_seq INCREMENT 1 MINVALUE 1 MAXVALUE 922337

CREATE TABLE "public"."usuarios"
(
    "is_deleted" boolean   DEFAULT false,
    "created_at" timestamp DEFAULT CURRENT_TIMESTAMP          NOT NULL
    "id"         bigint    DEFAULT nextval('usuarios_id_seq') NOT NULL
    "updated_at" timestamp DEFAULT CURRENT_TIMESTAMP          NOT NULL
    "apellidos"  character varying(255)                       NOT NULL
    "email"      character varying(255)                       NOT NULL
    "nombre"     character varying(255)                       NOT NULL
    "password"   character varying(255)                       NOT NULL
    "username"   character varying(255)                       NOT NULL
    CONSTRAINT "usuarios_email_key" UNIQUE ("email"),
    CONSTRAINT "usuarios_pkey" PRIMARY KEY ("id"),
    CONSTRAINT "usuarios_username_key" UNIQUE ("username")
) WITH (oids = false);

-- Contraseña: admin Admin1
-- Contraseña: user User1234
-- Contraseña: test test1234


INSERT INTO "usuarios" ("is_deleted", "created_at", "id", "updated_at"
                        "username")
VALUES ('f', '2023-11-02 11:43:24.724871', 1, '2023-11-02 11:43:24.724
        '$2a$10$vPaqZvZkz6jhb7U7k/V/v.5vprfNdOnh4sxi/qpPRkYTzPmFlI9p2'
       ('f', '2023-11-02 11:43:24.730431', 2, '2023-11-02 11:43:24.730
        '$2a$12$RUq2ScW1Kiizu5K4gKoK4OTz80.DWaruhdyfi2lZCB.KeuXTBh0S.'
       ('f', '2023-11-02 11:43:24.733552', 3, '2023-11-02 11:43:24.733
```

Carpeta / Archivo	Contenido o subcarpetas / archivos internos
	'$2a$10$Pd1yyq2NowcsDf4Cpf/ZXObYFkcycswqHAqBndE1wWJvYwRxlb.Pu' ('f', '2023-11-02 11:43:24.736674', 4, '2023-11-02 11:43:24.736 '$2a$12$3Q4.UZbvBMBEvIwwjGEjae/zrIr6S50NusUlBcCNmBd2382eyU0bS' CREATE TABLE "public"."categorias" ("is_deleted" boolean DEFAULT false, "created_at" timestamp DEFAULT CURRENT_TIMESTAMP NOT NULL, "updated_at" timestamp DEFAULT CURRENT_TIMESTAMP NOT NULL, "id" uuid NOT NULL, "nombre" character varying(255) NOT NULL, CONSTRAINT "categorias_nombre_key" UNIQUE ("nombre"), CONSTRAINT "categorias_pkey" PRIMARY KEY ("id")) WITH (oids = false); INSERT INTO "categorias" ("is_deleted", "created_at", "updated_at", "i VALUES ('f', '2023-11-02 11:43:24.717712', '2023-11-02 11:43:24.717712 'DEPORTES'), ('f', '2023-11-02 11:43:24.717712', '2023-11-02 11:43:24.717712 'COMIDA'), ('f', '2023-11-02 11:43:24.717712', '2023-11-02 11:43:24.717712 'BEBIDA'), ('f', '2023-11-02 11:43:24.717712', '2023-11-02 11:43:24.717712 'COMPLEMENTOS'), ('f', '2023-11-02 11:43:24.717712', '2023-11-02 11:43:24.717712 'OTROS'); ALTER TABLE ONLY "public"."productos" ADD CONSTRAINT "fk2fwq10nwymfv7fumctxt9vpgb" FOREIGN KEY (categori ALTER TABLE ONLY "public"."user_roles" ADD CONSTRAINT "fk2chxp26bnpqjibydrikgq4t9e" FOREIGN KEY (user_id)
src (o app)/	config/ (Ir al Punto 3 . Estructura Carpetas .Nivel src) models/ (Ir al Punto 3 . Estructura Carpetas .Nivel src) services/ (Ir al Punto 3 . Estructura Carpetas .Nivel src) uploads/ (Ir al Punto 3 . Estructura Carpetas .Nivel src) create.php delete.php details.php footer.php header.php index.php login.php logout.php tareas.php update-image.php

<table>
<tr><th>Carpeta / Archivo</th><th>Contenido o subcarpetas / archivos internos</th></tr>
<tr><td></td><td>update_image_file.php
update.php</td></tr>
<tr><td>tests/</td><td>(Para la realización de pruebas)</td></tr>
<tr><td>uploads/</td><td>favicon.ico (Lo crearás para tu aplicación)</td></tr>
<tr><td>vendor/</td><td>(Aquí se cargarán las dependecias , esta carpeta NO hace falta crearla , se generára
install)</td></tr>
<tr><td>.env</td><td>(Este archivo lo configuraremos de esta manera , lo deberás adaptar a tu base de da

<pre>
# .env file
POSTGRES_DB=tienda
POSTGRES_USER=admin
POSTGRES_PASSWORD=adminPassword123
POSTGRES_HOST=postgres-db
POSTGRES_PORT=5432
APP_PORT=8080
</pre></td></tr>
<tr><td>.gitignore

(para Github , No crear)</td><td><pre>
/vendor/
/uploads/
</pre></td></tr>
<tr><td>composer.json</td><td><pre>
{
  "autoload": {
    "psr-4": {
      "": "src/"
    }
  },
  "require-dev": {
    "phpunit/phpunit": "^9"
  },
  "require": {
    "vlucas/phpdotenv": "^5.6",
    "ramsey/uuid": "^4.7"
  }
}
</pre></td></tr>
</table>

Carpeta / Archivo	Contenido o subcarpetas / archivos internos
docker-compose.yml	

```yaml
version: '3.'

services:
  php-app:
    build: .
    container_name: php_app
    ports:
      - "${APP_PORT}:80"
    env_file: .env
    volumes:
      - .:/var/www/html # Monta toda la raíz del proyecto, incluido sr
    depends_on:
      - postgres-db
    environment:
      DATABASE_HOST: postgres-db
      DATABASE_USER: ${POSTGRES_USER}
      DATABASE_PASSWORD: ${POSTGRES_PASSWORD}
      DATABASE_NAME: ${POSTGRES_DB}
      APACHE_DOCUMENT_ROOT: /var/www/html/src # Así Apache busca index
    networks:
      - tienda-network

  postgres-db:
    image: postgres:12-alpine
    container_name: postgres_db
    env_file: .env
    ports:
      - "${POSTGRES_PORT}:5432"
    volumes:
      - ./database/init.sql:/docker-entrypoint-initdb.d/init.sql
      - db_data:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    networks:
      - tienda-network

  adminer:
    image: adminer
    container_name: adminer
    env_file: .env
    ports:
      - "8081:8080"
    depends_on:
      - postgres-db
    networks:
      - tienda-network
```

Carpeta / Archivo	Contenido o subcarpetas / archivos internos

```
volumes:
  db_data:
  uploads:


networks:
  tienda-network:
    driver: bridge
```

Dockerfile	

```
# Dockerfile
FROM php:8.0-apache

# Instala las extensiones pdo_pgsql para PHP y otras extensiones comun
RUN apt-get update && apt-get install -y --fix-missing \
        libpq-dev \
        libpng-dev \
        libonig-dev \
        libxml2-dev \
        zip \
        unzip \
        git \
    && apt-get clean \
    && docker-php-ext-install pdo_pgsql mbstring exif pcntl bcmath gd

# Instala Composer
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

# Configura el document root
ENV APACHE_DOCUMENT_ROOT /var/www/html/public

# Cambia el document root del servidor Apache
RUN sed -ri -e 's!/var/www/html!${APACHE_DOCUMENT_ROOT}!g' /etc/apache
RUN sed -ri -e 's!/var/www/!${APACHE_DOCUMENT_ROOT}!g' /etc/apache2/ap

# Habilita el mod_rewrite para Apache
RUN a2enmod rewrite
```

Readme.txt	Crea este archivo , donde deberás ir explicando como has desarrollado el proyecto y tus archivos a tu github.

...

2.3. Contenido de los Ficheros .Nivel src (o app)

Carpeta / Archivo	Contenido o subcarpetas / archivos internos
config/	**Config.php** Si usas otra base de datos que no sea postgress con PDO , tendrás que adaptar tu archivo

```php
<?php

namespace config;


use Dotenv\Dotenv;
use PDO;


class Config
{
    private static $instance;
    private $postgresDb;
    private $postgresUser;
    private $postgresPassword;
    private $postgresHost;
    private $postgresPort;
    private $db;

    private $rootPath = '/var/www/html/public/';
    private $uploadPath = '/var/www/html/public/uploads/';
    private $uploadUrl = 'http://localhost:8080/uploads/';

    private function __construct()
    {

        $dotenv = Dotenv::createImmutable($this->rootPath);
        $dotenv->load();

        // Cargar las variables de entorno y almacenarlas en las propiedade
        $this->postgresDb = getenv('POSTGRES_DB') ?? 'default_db';
        $this->postgresUser = getenv('POSTGRES_USER') ?? 'default_user';
        $this->postgresPassword = getenv('POSTGRES_PASSWORD') ?? 'default_p
        $this->postgresHost = getenv('POSTGRES_HOST') ?? 'localhost';
        $this->postgresPort = getenv('POSTGRES_PORT') ?? '5432';
        $this->db = new PDO("pgsql:host={$this->postgresHost};port={$this->
    }

    public static function getInstance(): Config
    {
```

Carpeta / Archivo	Contenido o subcarpetas / archivos internos

```
        if (!isset(self::$instance)) {
            self::$instance = new Config();
        }
        return self::$instance;
    }


    // Magic methos for get and set
    public function __get($name)
    {
        return $this->$name;
    }

    public function __set($name, $value)
    {
        $this->$name = $value;
    }

}
```

models/	**Categoria.php** (Te pongo aquí un ejemplo de categorías) . Deberas implementar tu **Prod**

```
<?php

namespace models;

use Ramsey\Uuid\Uuid;

class Categoria
{
    private $id;
    private $nombre;
    private $createdAt;
    private $updatedAt;
    private $isDeleted;

    public function __construct($id = null, $nombre = null, $createdAt = nu
    {
        $this->id = $id;
        $this->nombre = $nombre;
        $this->createdAt = $createdAt;
        $this->updatedAt = $updatedAt;
        $this->isDeleted = $isDeleted;
    }

    public function getId()
```

Carpeta / Archivo	Contenido o subcarpetas / archivos internos

```
    {
        return $this->id;
    }

    // Magic method for get and set
    public function __get($name)
    {
        return $this->$name;
    }

    public function __set($name, $value)
    {
        $this->$name = $value;
    }

    private function generateUUID()
    {
        return Uuid::uuid4()->toString();
    }
}

?>
```

services/	**Categoriasservice.php** (Te pongo un ejemplo de Categoriasservice.php , tu deberás impl

```
<?php

namespace services;

use models\Categoria;
use PDO;

require_once __DIR__ . '/../models/Categoria.php';

class CategoriasService
{
    private $pdo;

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

    public function findAll()
    {
        $stmt = $this->pdo->prepare("SELECT * FROM categorias ORDER BY id A
```

Carpeta / Archivo	Contenido o subcarpetas / archivos internos

```
        $stmt->execute();

        $categorias = [];
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $categoria = new Categoria(
                $row['id'],
                $row['nombre'],
                $row['created_at'],
                $row['updated_at'],
                $row['is_deleted']
            );
            $categorias[] = $categoria;
        }
        return $categorias;
    }

    public function findByName($name)
    {
        $stmt = $this->pdo->prepare("SELECT * FROM categorias WHERE nombre
        $stmt->execute(['nombre' => $name]);

        $row = $stmt->fetch(PDO::FETCH_ASSOC);
        if (!$row) {
            return false;
        }
        $categoria = new Categoria(
            $row['id'],
            $row['nombre'],
            $row['created_at'],
            $row['updated_at'],
            $row['is_deleted']
        );
        return $categoria;
    }
}
```

Productosservice.php -Implementar el resto :

```
<?php

namespace services;

use models\Producto;
use PDO;
use Ramsey\Uuid\Uuid;
```

Carpeta / Archivo	Contenido o subcarpetas / archivos internos

```php
require_once __DIR__ . '/../models/Producto.php';

class ProductosService .........
```

Sessionservice.php-Implementar el resto

```php
<?php

namespace services;

/**
 * Class SessionService
 * @package services
 * Esta clase se encarga de gestionar la sesión de usuario
 */
class SessionService
{.......................................
```

Userservice.php-Implementar el resto

```php
<?php

namespace services;


use Exception;
use models\User;
use PDO;

require_once __DIR__ . '/../models/User.php';

class UsersService
{
    private $db;

    public function __construct(PDO $db)
    {
        $this->db = $db;
    }


    public function authenticate($username, $password): User .............
```

...

...

2.4. Tabla de rutas

Tabla de archivos y sus includes recomendados para tus clases (Las utilizaras más tarde)

Archivo	Rutas de `require_once` necesarias
public/index.php	`../vendor/autoload.php`, `../app/header.php`, `../app/footer.php`, `../app/services/SessionService.php`, `../app/config/Config.php`, `../app/services/ProductosService.php`, `../app/services/CategoriasService.php`
app/login.php	`../vendor/autoload.php`, `SessionService.php`, `UsersService.php`, `../config/Config.php`
app/logout.php	`../vendor/autoload.php`, `SessionService.php`
app/header.php	`../vendor/autoload.php`, `services/SessionService.php`
app/footer.php	`../vendor/autoload.php`, opcional `services/SessionService.php`
app/create.php	`../vendor/autoload.php`, `header.php`, `footer.php`, `services/SessionService.php`, `config/Config.php`, `services/ProductosService.php`, `services/CategoriasService.php`
app/update.php	`../vendor/autoload.php`, `header.php`, `footer.php`, `services/SessionService.php`, `config/Config.php`, `services/ProductosService.php`, `services/CategoriasService.php`
app/update-image.php	`../vendor/autoload.php`, `header.php`, `footer.php`, `services/SessionService.php`, `config/Config.php`, `services/ProductosService.php`
app/update_image_file.php	`../vendor/autoload.php`, `header.php`, `footer.php`, `services/SessionService.php`, `config/Config.php`, `services/ProductosService.php`
app/delete.php	`../vendor/autoload.php`, `header.php`, `footer.php`, `services/SessionService.php`, `config/Config.php`, `services/ProductosService.php`
app/details.php	`../vendor/autoload.php`, `header.php`, `footer.php`, `services/SessionService.php`, `config/Config.php`, `services/ProductosService.php`, `services/CategoriasService.php`

4.Configuración del entorno de trabajo

Nota : Nosotros utilizaremos Docker , pero puedes utilizar Xamp , Laragon , etc...

Pautas guiadas para montar un proyecto PHP con Docker

(Si tu ya tienes tus archivos creados del punto 2 , ves al paso 4)

1. **Prepara la estructura del proyecto**
   - Organiza las carpetas del proyecto siguiendo una estructura recomendada: `/src`, `/database`, `/images`, `/config`, etc.
   - Coloca todos los archivos PHP (como `index.php`, `header.php`, etc.) en la carpeta `/src`.
   - Añade los archivos de configuración necesarios (`composer.json`, `.env`, etc.) en la raíz del proyecto.
2. **Prepara el archivo Dockerfile**
   - El `Dockerfile` define la imagen del contenedor PHP + Apache.
   - Debe especificar la versión de PHP y copiar archivos al contenedor.
   - Ejemplo básico:

```
FROM php:8.2-apache
COPY src/ /var/www/html/
COPY --from=composer /usr/bin/composer /usr/bin/composer
WORKDIR /var/www/html
```

3. **Prepara el archivo docker-compose.yml**
   - Define los servicios (PHP, base de datos, adminer, etc.).
   - Monta el directorio local como volumen en el contenedor.
   - Ejemplo simple:

```
services:
  php-app:
    build: .
    ports:
      - "8080:80"
    volumes:
      - ./src:/var/www/html
    environment:
      APACHE_DOCUMENT_ROOT: /var/www/html
  db:
    image: postgres:12-alpine
    environment:
      POSTGRES_DB: testdb
      POSTGRES_USER: testuser
      POSTGRES_PASSWORD: testpass
    ports:
      - "5432:5432"
    volumes:
      - ./database:/docker-entrypoint-initdb.d/
```

4. **Prepara el entorno**
   Antes de lanzar Docker, instala las dependencias ejecutando `composer install` en tu máquina local para que la carpeta `vendor/` esté completa.
5. **Inicia los contenedores**
   - Desde la raíz del proyecto, ejecuta:

     ```
     docker-compose up -d --build
     ```

     Esto descargará las imágenes necesarias y montará el entorno.
6. **Comprueba que funciona**
   - Accede a tu proyecto desde el navegador: **http://localhost:8080 <http://localhost:8080>** .
   - Adminer estará disponible (si lo has incluido) en otro puerto.
7. **Gestión y cierre**
   - Para parar los servicios usa:

     ```
     docker-compose down
     ```

   - Para ver los logs de un servicio:

     ```
     docker-compose logs php-app
     ```

Consejos :

- Repasa la estructura de carpetas antes de lanzar Docker.
- Usa rutas relativas correctas para los `require_once` en PHP.
- Comprueba los permisos si aparece un error "Forbidden".
- Consulta los logs de los contenedores ante cualquier error.

4.1. Explicación general entorno (extra)

1. Composer

Composer es el **gestor de dependencias de PHP**. Permite instalar y actualizar librerías externas, gestionar versiones y generar `composer.json` y `composer.lock`.

Comandos comunes

```
composer install
```

```
composer require vlucas/phpdotenv composer update
```

```
Al ejecutar composer install se crea la carpeta vendor/ con las dependencias y autoload.php.
```

2. Carpeta `vendor/`

Carpeta generada por Composer que contiene todas las librerías instaladas y el archivo `autoload.php`. No debe modificarse manualmente y normalmente se añade a `.gitignore`.

3. Archivo `.env`

Archivo de texto que almacena **variables de entorno** (configuración sensible: contraseñas, host, puertos, modos). Se mantiene fuera del código fuente para facilitar despliegues en distintos entornos.

```
APP_ENV=development
```

```
DB_HOST=localhost DB_USER=root DB_PASS=secret
```

4. Dotenv

Es la librería (por ejemplo `vlucas/phpdotenv`) que **lee el .env** y carga sus valores en variables de entorno accesibles desde PHP (`$_ENV` o `getenv()`).

```
// Ejemplo en PHP
```

```
$dotenv = Dotenv\Dotenv::createImmutable(DIR); $dotenv->load(); echo $_ENV['DB_HOST'];
```

5. Dockerfile

Archivo que describe cómo construir una **imagen Docker** (es la receta). Define la imagen base, copia archivos, instala extensiones y deja el sistema listo para ejecutar la aplicación.

```
FROM php:8.2-apache
```

```
COPY . /var/www/html RUN docker-php-ext-install pdo pdo_mysql
```

```
Cada instrucción crea una capa en la imagen; usar .dockerignore para excluir archivos que no d
```

6. `docker-compose.yml`

Archivo que define y orquesta **varios servicios** (contenedores) y cómo se conectan: redes, volúmenes, variables de entorno. Facilita levantar todo el entorno con un solo comando.

```
version: '3.8'


services: php-app: build: . ports: - "8080:80" volumes: - .:/var/www/html db: image: mysql:8 e

Relación: Dockerfile = receta de cada imagen. docker-compose = cómo conectar y ejecutar varias
```

7. Namespaces

Los **namespaces** organizan clases PHP en espacios de nombres para evitar colisiones y facilitar el autoload de Composer.

```
namespace Models;


class User {}

/* uso */ use Models\User; $user = new User();

En composer.json se define el mapeo psr-4 para que Composer genere el autoload correcto.
```

8. Comandos Docker (útiles)

Fuera del contenedor (host):

```
docker-compose up -d      # levantar servicios en segundo plano


docker-compose down # apagar y eliminar redes/volúmenes creados por compose docker ps # listar
```

Entrar dentro de un contenedor (para ejecutar comandos como si fueses servidor):

```
docker exec -it nombre_contenedor bash


una vez dentro:

php -v composer install ls -la /var/www/html
```

En resumen visual

Elemento	Función principal	Ejemplo / relación
Composer	Gestor de dependencias PHP	`composer install`
`vendor/`	Carpeta con librerías instaladas	Contiene `autoload.php`
`.env`	Variables de entorno y configuración sensible	`DB_HOST=localhost`
Dotenv	Carga `.env` en variables de entorno	`Dotenv::createImmutable()`
Dockerfile	Construye la imagen (receta)	`FROM php:8.2-apache`

Elemento	Función principal	Ejemplo / relación
`docker-compose.yml`	Orquesta servicios (PHP, MySQL...)	Levantar varios contenedores juntos
Namespaces	Organización de clases PHP	`namespace Models;`
Comandos Docker	Gestiona contenedores, imágenes, etc.	`docker compose up`, `docker exec`

5.Patrón Singleton y como aplicarlo al CRUD

¿Qué es el patrón Singleton y cómo se aplica en este CRUD?

El **patrón Singleton** es un patrón de diseño que garantiza que una clase tiene **una única instancia** y proporciona un punto de acceso global a dicha instancia. Es útil cuando necesitas compartir un único recurso por toda tu aplicación, evitando duplicados innecesarios y controlando el acceso.

¿Dónde se usa Singleton en esta aplicación?

- **SessionService**
  Gestiona todos los datos de sesión del usuario (login, roles, contadores, etc.) usando el método `getInstance()` para obtener la única instancia operativa en cada petición. Así, los datos del usuario y las operaciones con la sesión son coherentes y seguras en todos los puntos del código.
- **Config**
  Centraliza la configuración del sistema (acceso a la base de datos, rutas de uploads, etc.) garantizando que toda la aplicación utilice el mismo conjunto de valores mediante `Config::getInstance()`. Esto previene inconsistencias y facilita modificaciones globales en la configuración.

¿Cómo funciona su implementación en los archivos?

- Cada clase Singleton tiene:
  - Un atributo privado y estático que almacena la instancia única.
  - Un método estático (`getInstance()`) que crea la instancia la primera vez y la devuelve siempre.
  - El constructor privado, impidiendo crear objetos directamente con `new`, así solo se puede acceder por `getInstance()`.
- Ejemplo real usado en la aplicación:

```
$session = SessionService::getInstance();
$config = Config::getInstance();
```

- Así, cualquier operación de sesión (login, logout, control de permisos) y de configuración usa el mismo objeto en todas las partes del proyecto.

Ventajas principales para el desarrollo:

- Evita duplicidad de datos y recursos.
- Centraliza la gestión y simplifica el mantenimiento.
- Facilita la seguridad y el control de estado único en sesión y configuración.

6. Archivos carpeta src

6.1. index.php

Explicación del archivo principal del CRUD de productos (index.php)

- **Carga de dependencias**
  El archivo comienza incluyendo las librerías y clases necesarias del proyecto y de Composer (`vendor/autoload.php`), además de los archivos de servicios y modelos propios.
- **Gestión de sesión y configuración**
  Inicializa la sesión de usuario y la configuración principal usando los servicios `SessionService` y `Config`.
- **Interfaz principal**
  - Incluye el archivo común `header.php`.
  - Muestra un mensaje personalizado de bienvenida usando los datos de la sesión.
  - Presenta un formulario de búsqueda que permite filtrar productos por marca o modelo.
- **Tabla de productos**
  - Recupera y muestra todos los productos (o filtrados) en una tabla.
  - Cada fila muestra los datos clave y una imagen del producto.
  - Se incluyen botones para ver detalles, editar, cambiar imagen y eliminar, enlazando a los respectivos scripts.
- **Acciones adicionales**
  - Botón destacado para crear un nuevo producto.
  - Si el usuario ha iniciado sesión, muestra el número de visitas y la fecha de último login en la parte inferior.
- **Pie de página y scripts**
  - Finaliza incluyendo el `footer.php`, además de los scripts de Bootstrap y jQuery para ofrecer una experiencia visual moderna.

Resumen del funcionamiento:
Este archivo es el centro del proyecto CRUD de productos: permite gestionar (ver, crear, buscar, editar, eliminar) todos los productos desde una interfaz moderna y sencilla, mostrando información personalizada para cada usuario.

6.2. header.php

Explicación del archivo de cabecera (header.php)

- **Gestión de sesión**
  El archivo carga el servicio de sesiones (`SessionService`) para identificar si el usuario ha iniciado sesión y determinar su nombre o mostrar "Invitado".
- **Barra de navegación**
  Usa Bootstrap para mostrar una barra de navegación atractiva y moderna. Incluye el logo, nombre del proyecto y varias opciones de menú:
  - Bóton "Login" o "Logout" que cambia según el estado del usuario.
  - Enlace a la página de productos (`index.php`).
  - Enlace para crear un nuevo producto.
  - El nombre de usuario (o "Invitado") aparece alineado a la derecha.
- **Accesibilidad y mobile-friendly**
  El menú es responsive, adaptándose a móviles gracias a los botones de toggling de Bootstrap.

Resumen:
El **header.php** es reutilizado en todas las páginas para mostrar un menú coherente, cambiar dinámicamente según la sesión y permitir al usuario navegar, iniciar/cerrar sesión y ver su nombre en cada vista principal del proyecto.

6.3. footer.php

Explicación del archivo de pie de página (footer.php)

- **Finalización visual de la página**
  El pie de página se muestra alineado al centro y separado visualmente mediante una línea (`<hr>`).
- **Información del proyecto y autoría**
  Indica que se trata de un CRUD de productos para el módulo de 2º DAW, realizado por (ejemplo :Eva María Gómez Abad en el IES Juan De Garay).
- **Enlace externo**
  Proporciona un enlace clicable a Github del autor , abriéndose en nueva pestaña.

Resumen:
El **footer.php** completa todas las páginas del proyecto, aportando información institucional y de autoría, y mejorando la presentación general del sitio CRUD.

6.4. create.php

Explicación del archivo de creación de productos (create.php)

- **Control de acceso**
  Solo los usuarios tipo "admin" pueden acceder. Si no tienes permisos, el sistema muestra un mensaje y redirige al índice.
- **Servicios y dependencias**
  El archivo carga todos los servicios y modelos necesarios, especialmente para acceder a la gestión de productos y categorías.
- **Formulario de alta**
  Presenta un formulario con los campos básicos del producto: Marca, Modelo, Descripción, Precio, Categoría, Stock, Imagen. (Por ejemplo)
  - Cada campo se valida y muestra mensajes de error en caso necesario.
  - La categoría se selecciona de un desplegable dinámico que se carga desde la base de datos.
- **Procesamiento y validación**
  - Al enviar el formulario, los datos se filtran, validan y comprueban.
  - Si hay errores, se muestran junto al campo correspondiente.
  - Si todo está bien, se crea el producto en la base de datos y se muestra un mensaje de éxito.
- **Enlaces y navegación**
  Incluye enlaces para volver al índice y usabilidad con Bootstrap.

Resumen:
create.php controla tanto el acceso como la lógica y la presentación del formulario de alta de productos. Valida los datos, realiza el alta y orienta la navegación del usuario, garantizando que sólo los administradores gestionen el catálogo de productos.

6.5. update.php

Explicación del archivo de actualización de productos (update.php)

- **Control de permisos**
  Solo los usuarios con rol "admin" pueden acceder. Si no tienes permisos, el sistema muestra un mensaje y te redirige al índice.
- **Servicios y dependencias**
  Carga todos los servicios, modelos y configuración necesarios para gestionar productos y categorías.
- **Consulta y validación inicial**
  Obtiene el ID del producto a editar desde la URL (`GET`) y busca el producto en la base de datos. Si no encuentra el producto, redirige automáticamente.
- **Formulario de edición**
  El formulario muestra todos los campos editables del producto, pre-rellenados con sus valores actuales: Marca, Modelo, Descripción, Precio, Stock, Imagen (solo lectura) y Categoría.(Por ejemplo)
  La selección de categoría se realiza en un desplegable dinámico cargado desde la base de datos.
- **Procesamiento del formulario**
  Al enviar el formulario (`POST`):
  - Se filtran y validan todos los datos.
  - Se muestran los errores junto al campo correspondiente en caso de que falte o no sea válido algún dato.
  - Si es correcto, se actualiza el producto en la base de datos y se notifica al usuario con un mensaje de éxito.
- **Diseño y navegación**
  Toda la estructura utiliza Bootstrap para una presentación moderna, incluye cabecera y pie comunes y botones para facilitar la navegación.

Resumen:
El **update.php** permite a los administradores modificar cualquier producto de la tienda, validando los datos y manteniendo la seguridad y la usabilidad de la aplicación.

6.6. delete.php

Explicación del archivo de eliminación de productos (delete.php)

- **Control de permisos**
  Solo los usuarios administradores pueden eliminar productos. Si no tienes permisos, se muestra un mensaje y te redirige al índice.
- **Recepción del parámetro**
  El identificador (`id`) del producto a eliminar se recibe por la URL. Si no es válido, se redirige al índice.
- **Eliminación de la imagen asociada**
  Antes de borrar el producto, si este tiene una imagen personalizada, se elimina físicamente del servidor para evitar archivos huérfanos.
- **Eliminación del producto**
  Una vez gestionada la imagen, se borra el producto de la base de datos usando el método adecuado del servicio correspondiente.
- **Notificación y redirección**
  Cuando se completa la acción, se muestra un mensaje de confirmación al usuario y se redirige al índice.

Resumen:
delete.php se encarga de gestionar de forma segura la eliminación de productos y sus imágenes, limitando el acceso a usuarios administradores y garantizando una correcta limpieza de recursos.

6.7. details.php

Explicación del archivo de detalles de producto (details.php)

- **Recepción y validación de parámetros**
  Se obtiene el `id` del producto a mostrar desde la URL. Si no es válido o no existe el producto, se redirige al índice para evitar errores.
- **Carga de datos**
  Se cargan las dependencias y servicios necesarios, y se busca en la base de datos el producto correspondiente usando el identificador recibido.
- **Presentación de la información**
  Se muestran, de forma estructurada y clara, todos los datos del producto (ID, marca, modelo, descripción, precio, imagen, stock y categoría por ejemplo) en una lista (definición) formateada con Bootstrap.
- **Navegación básica**
  Se incluye un botón para volver al listado general de productos. La página hereda cabecera y pie para mantener la coherencia visual en toda la aplicación.

Resumen:
El archivo **details.php** muestra todos los datos de un producto seleccionado de manera ordenada y visual, controlando que siempre se consulta un producto válido y guiando al usuario para volver al listado principal.

6.8. login.php

Explicación del archivo de autenticación (login.php)

- **Inicio de sesión y validación**
  Muestra un formulario donde el usuario introduce su nombre y contraseña para acceder al sistema de gestión de productos.
- **Gestión de entrada y errores**
  Al enviar el formulario, los datos se filtran y validan.
  Si falta alguno, o no coinciden con ningún usuario registrado, se muestra un mensaje de error claro.
- **Autenticación y control de rol**
  Intenta autenticar al usuario con el servicio adecuado (`UsersService`). Si la autenticación es correcta:
  - Establece la sesión con el usuario (y, si es administrador, lo marca).
  - Redirige automáticamente al usuario al índice (portada de productos).

  Si la autenticación falla, muestra error y permanece en login.
- **Interfaz y diseño**
  Utiliza Bootstrap para una apariencia clara y centrada. Incluye la cabecera común con logo y nombre del proyecto.
- **Pie y scripts**
  Incluye el pie de página estándar y scripts necesarios para una presentación moderna y responsive.

Resumen:
El archivo **login.php** gestiona el acceso seguro a la aplicación: valida el usuario, diferencia entre roles y muestra mensajes de error claros si la autenticación falla.

6.9.logout.php

Explicación del archivo de cierre de sesión (logout.php)

- **Servicio de sesión**
  El archivo carga el servicio responsable de la gestión de la sesión de usuario (`SessionService`).
- **Proceso de logout**
  Llama al método `logout()` para cerrar por completo la sesión del usuario activo, eliminando todos los datos de sesión.
- **Redirección automática**
  Una vez finalizada la sesión, redirige automáticamente al usuario al índice, mostrando la portada como "Invitado".

Resumen:
El archivo **logout.php** permite a cualquier usuario cerrar su sesión de forma segura y volver al inicio de la aplicación, garantizando que sus datos privados quedan protegidos y bloqueados tras el logout.

6.10. update_image

Explicación del archivo de formulario de actualización de imagen (update-image.php)

- **Control de permisos**
  Sólo los usuarios con rol "admin" pueden acceder. Si no tienes permisos, el sistema muestra un mensaje y te redirige al índice.
- **Validación de entrada**
  Obtiene el `id` del producto por la URL y comprueba que el producto existe en la base de datos. Si no existe, te avisa y te redirige.
- **Muestra datos y formulario**
  Visualiza los datos esenciales del producto: ID, marca, modelo e imagen actual.
  - Incluye un formulario para subir una nueva imagen (acepta solo files tipo imagen).
  - Al enviar el formulario, los datos y archivos se gestionan en `update_image_file.php`.
- **Usabilidad y navegación**
  Incluye botones para actualizar la imagen y volver al listado principal. Utiliza Bootstrap para diseño y experiencia de usuario.
- **Cabecera y pie**
  El archivo utiliza `header.php` y `footer.php` para cohesión visual.

Resumen:
update-image.php permite que los administradores seleccionen y suban una nueva imagen para el producto, garantizando la validación y la seguridad en la operación.

6.11. update_image_file

Explicación del archivo de actualización de imagen del producto (update-image_file.php)

- **Gestión del envío del formulario**
  El archivo sólo trabaja si la petición se realiza mediante POST y se incluye correctamente el archivo de imagen.
- **Validación y comprobación de datos**
  - Recibe el `id` del producto y valida que exista.
  - Comprueba que el archivo subido es una imagen (JPEG o PNG), tanto por MIME-type como por extensión.
  - Si no es válido, redirige al índice directamente.
- **Procesamiento de la imagen**
  - Genera el nuevo nombre de imagen a partir del uuid del producto y la extensión correspondiente.
  - Mueve el archivo físico de la ruta temporal a la carpeta de uploads definida en la configuración del proyecto.
- **Actualización en base de datos**
  - Actualiza la ruta de la imagen en el producto correspondiente.
  - Guarda el cambio en la base de datos.
- **Redirección y control de flujo**
  - Si todo funciona, redirige a la página para actualizar la imagen.
  - Si hay cualquier error, redirige al índice general.

Resumen:
El archivo **update-image_file.php** permite modificar la imagen de los productos de forma segura, validando extensiones, tipos y gestionando tanto archivos físicos como rutas en la base de datos.

7. Archivos carpeta services

7.1. Productosservices.php

Explicación de la clase de servicio ProductosService.php

- **Propósito de la clase**
  ProductosService es la clase central que gestiona todas las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los productos en la base de datos. Actúa como intermediario entre el código de aplicación y la base de datos.
- **Constructor y conexión**
  Recibe una conexión PDO en el constructor para poder realizar todas las consultas a la base de datos usando prepared statements (consultas preparadas) para mayor seguridad.
- **Principales métodos y funcionalidades**
  - **findAllWithCategoryName($searchTerm)**:
    - Recupera todos los productos con su categoría asociada mediante JOIN.
    - Si se proporciona término de búsqueda, filtra por marca o modelo.
    - Devuelve un array de objetos `Producto`.
  - **findById($id)**:
    - Busca un producto específico por su ID.
    - Incluye la información de la categoría asociada.
    - Devuelve `null` si no lo encuentra o un objeto `Producto`.
  - **save(Producto $producto)**:
    - Inserta un nuevo producto en la base de datos.
    - Genera automáticamente un UUID único y asigna imagen por defecto.
    - Establece fechas de creación y actualización automáticamente.
  - **update(Producto $producto)**:
    - Actualiza un producto existente por su ID.
    - Modifica todos los campos editables y actualiza la fecha de modificación.
  - **deleteById($id)**:
    - Elimina físicamente un producto de la base de datos por su ID.
    - Devuelve `true` si la operación fue exitosa.
- **Características de seguridad**
  - Usa prepared statements para evitar inyecciones SQL.
  - Vincula parámetros con tipos específicos (INT, STR) para mayor control.
  - Genera UUIDs únicos usando la librería Ramsey/UUID.

Resumen:
La clase **ProductosService** centraliza toda la lógica de gestión de productos, ofreciendo métodos seguros y eficientes para las operaciones CRUD, incluyendo búsquedas, relaciones con categorías y manejo automático de metadatos (fechas, UUIDs).

7.2. CategoriasServices.php

Explicación de la clase de servicio CategoriasService.php

- **Propósito de la clase**
  CategoriasService es una clase que actúa como intermediaria entre la base de datos y el resto de la aplicación para todo lo relacionado con las categorías de productos.
  Centraliza la lógica relacionada con el acceso y tratamiento de datos de la tabla `categorias`.
- **Gestión de la conexión**
  Recibe la conexión PDO en el constructor y la almacena, lo que permite reutilizar la conexión a la base de datos durante toda la vida útil de la instancia.
- **Principales métodos**
  - **findAll()**: Recupera todas las categorías ordenadas por su ID.
    - Ejecuta la consulta.
    - Crea objetos `Categoria` para cada fila y los retorna como array.
  - **findByName($name)**: Busca una categoría por su nombre exacto.
    - Devuelve `false` si no la encuentra, o bien una instancia de `Categoria` si existe.
- **Uso típico en la aplicación**

  ```
  $config = Config::getInstance();

  $categoriasService = new CategoriasService($config->db);

  $categorias = $categoriasService->findAll();
  ```

  Así, cualquier script PHP puede recuperar categorías, comprobar si ya existen, o asociar productos a ellas reutilizando siempre la misma lógica.

Resumen:
La clase **CategoriasService** facilita la gestión profesional de categorías de producto, centralizando toda la lógica de consulta y evitando duplicidad de código en la aplicación.

7.3. SessionService.php

Explicación de la clase SessionService.php

- **Propósito**
  SessionService centraliza toda la lógica relacionada con la sesión de usuario: inicio, cierre, autenticación, control de visitas y roles. Permite una gestión profesional y segura del estado de cada usuario en la aplicación.
- **Patrón Singleton aplicado**
  Para garantizar que sólo se use una instancia en cada petición PHP, la clase implementa el patrón Singleton:
  - El método `getInstance()` devuelve siempre la única instancia disponible.
  - El constructor es privado, impidiendo la creación de más objetos.
- **Gestión y seguridad de la sesión**
  - **Expiración automática**: La sesión se considera caducada tras una hora de inactividad (odede>$expireAfterSeconds).
  - **Inicialización**: Define y actualiza variables clave (odede>loggedIn, <

7.4. UsersService.php

Explicación de la clase de servicio UsersService.php

- **Propósito de la clase**
  UsersService gestiona toda la lógica para la autenticación de usuarios y la consulta de datos relacionados con los usuarios registrados en el sistema.
- **Conexión a la base de datos**
  Recibe una instancia de `PDO` en su constructor para ejecutar consultas seguras y eficientes sobre la base de datos.
- **Principales métodos**
  - **authenticate($username, $password)**:
    - Busca el usuario por nombre utilizando `findUserByUsername()`.
    - Verifica la contraseña enviada usando `password_verify()`, que compara con la contraseña almacenada en formato hash seguro (bcrypt).
    - Si el usuario existe y la contraseña es correcta, devuelve el objeto `User`.
    - Si no es correcto, lanza una excepción y se muestra error en el login.
  - **findUserByUsername($username)**:
    - Realiza una consulta para encontrar el usuario por su nombre.
    - Recupera también los roles asociados al usuario de una tabla separada (`user_roles`).
    - Devuelve una instancia `User` con todos los datos relevantes o `null` si no existe.
- **Modelo User y roles**
  El resultado es siempre un objeto odede>User que incluye todos los campos del usuario y un array con los roles (ejemplo: ADMIN, USER, etc.), lo que permite gestionar los permisos y las vistas de la aplicación.
- **Seguridad y buenas prácticas**
  - Utiliza `password_verify()` para validar contraseñas seguras.
  - Usa consultas preparadas (`prepare` y odede>bindParam) para evitar inyecciones SQL.
  - Maneja roles para distinguir usuarios normales y administradores.

Resumen:
La clase **UsersService** permite autenticar usuarios y asociar roles, base crucial para controlar el acceso, los permisos y la seguridad en la aplicación de gestión CRUD.

8. Archivos carpeta models

8.1. Categoria.php

Explicación de la clase modelo Categoria

- **Propósito de la clase**
  Categoria es una clase modelo que representa una categoría de producto en el sistema. Sus instancias contienen los datos necesarios para manipular categorías en la aplicación y en la base de datos.
- **Atributos principales**
  - `id`: Identificador único de la categoría.
  - `nombre`: Nombre de la categoría.
  - `createdAt`, `updatedAt`: Fechas de creación y última actualización.
  - `isDeleted`: Marca lógica de borrado o desactivación.
- **Métodos clave**
  - **__construct()**: Permite inicializar todos los campos de la categoría al crear una nueva instancia con valores opcionales.
  - **getId()**: Devuelve el ID de la categoría.
  - **__get($name) / __set($name, $value)**: Métodos mágicos que facilitan el acceso y la modificación dinámica de atributos (propiedades) del objeto.
  - **generateUUID()**: Método privado que genera un identificador único (UUID) utilizando la librería Ramsey/Uuid; puede usarse para identificadores irrepetibles si se necesita.
- **Uso dentro de la aplicación**
  Las instancias de **Categoria** son utilizadas por los servicios (por ejemplo, `CategoriasService`) para transferir y manipular datos de categorías, mostrando información o realizando operaciones sobre ellas (buscar, listar, asignar a productos, etc.).

Resumen:
La clase **Categoria** sustenta las operaciones con categorías de producto, centralizando su estructura de datos y facilitando el acceso a sus propiedades en todo el sistema CRUD.

8.2. Producto.php

Explicación de la clase modelo Producto.php

- **Propósito de la clase**
  Producto es una clase modelo que representa cada producto del catálogo en la aplicación. Permite crear, gestionar y transferir objetos producto entre los diferentes servicios y capas.
- **Atributos principales**
  - `id`: Identificador único del producto.
  - `uuid`: Identificador universal único.
  - odede>descripcion: Descripción textual del producto.
  - `imagen`: URL de la imagen del producto (usa `$IMAGEN_DEFAULT` si no hay una personalizada).
  - `marca`, odede>modelo, `precio`, `stock`: Datos básicos del producto.
  - `createdAt`, `updatedAt`: Fechas de alta y última modificación.
  - `categoriaId`: ID de la categoría asociada.
  - `categoriaNombre`: Nombre de la categoría (para facilitar la consulta y visualización).
  - `isDeleted`: Marca lógica para manejar borrados.
- **Constructor flexible**
  Permite crear nuevos objetos **Producto** con todos los parámetros principales, dando flexibilidad cuando se recibe información de la base de datos o se crea manualmente.
- **Métodos mág**

8.3. User.php

Explicación de la clase modelo User,php

- **Propósito de la clase**
  User es la clase que representa a cada usuario del sistema. Gestiona tanto los datos básicos del usuario como sus privilegios (roles).
- **Atributos principales**
  - `id`: Identificador único del usuario.
  - `username`: Nombre de usuario para login.
  - `password`: Contraseña (encriptada con bcrypt).
  - `nombre`, `apellidos`, `email`: Datos personales.
  - `createdAt`, `updatedAt`: Fechas de registro y última modificación.
  - `isDeleted`: Marca lógica de borrado o desactivación.
  - `roles`: Array con los roles del usuario (por ejemplo, 'ADMIN', 'USER').
- **Constructor**
  Permite crear un objeto usuario con toda su información y roles, facilitando la autenticación y la gestión de privilegios.
- **Métodos mágicos de acceso**
  - `__get($name)` y `__set($name, $value)`: Permiten el acceso dinámico y modificación de cualquier atributo, simplificando la reutilización en servicios y formularios.
- **Usos principales en la aplicación**
  - Gestión de autenticación y login.
  - Gestión de roles y privilegios diferenciando administradores y usuarios normales.
  - Transferencia de datos entre consultas SQL y servicios, especialmente con `UsersService`.

Resumen:
La clase **User** facilita la gestión completa de los datos y roles de los usuarios en el CRUD, permitiendo operaciones seguras y estructuradas acorde al modelo orientado a objetos.

9. Archivos PDF BBDD y PDO

9.1. BBDD

Lee con atención este PDF sobre BBDD , ta ayudará en tú proyecto

- Acceso a BBDD.pdf (Ventana nueva)

9.2. PDO

Lee con atención este PDF sobre la conexión con PDO , ta ayudará en tú proyecto

- Introducción a PDO.pdf (Ventana nueva)