Documentación Técnica - TechZone

Libs

La carpeta libs contiene tres archivos con las funciones auxiliares reutilizables que se encargan de dar el soporte a la autenticación mediante JSON Web Tokens (JWT) y a varias funcionalidades del lado del cliente.

jwt.utils.php

El archivo jwt.utils.php incluye un conjunto de funciones encargadas de la gestión de autenticación basada en JWT desde el lado servidor. Su objetivo es la creación, validación y decodificación segura de los tokens que engloban la información sobre la sesión del usuario.

base64url_encode(string $data) y base64url_encode(string $data)

```php
function base64url_encode(string $data) {
    return rtrim(strtr(base64_encode($data), '+/', '-_'), '=');
}

function base64url_decode(string $data) {
    $remainder = strlen($data) % 4;
    if ($remainder) {
        $data .= str_repeat('=', 4 - $remainder);
    }
    return base64_decode(strtr($data, '-_', '+/'));
}
```

Imagen de codificación y decodificación

Implementa unas funciones auxiliares para codificar y decodificar datos en Base64 URL, adaptando la codificación para un uso seguro en URLs y eliminando los caracteres especiales.

generateJWT(array $payload, string $clave)

```php
function generateJWT(array $payload, string $clave) {
    $header = base64url_encode(json_encode(['alg' => 'HS256', 'typ' => 'JWT']));
    $body   = base64url_encode(json_encode($payload));
    $signature = hash_hmac('sha256', "$header.$body", $clave, true);
    $signature = base64url_encode($signature);

    return "$header.$body.$signature";
}
```

Imagen de la generación del token

Proporciona una función que genera el token JWT, construyendo el header y el body en JSON, codificándolos y firmándolos con HMAC SHA-256 para garantizar la integridad del token.

validateJWT(string $jwt, string $clave)

```php
function validateJWT(string $jwt, string $clave) {
    $partes = explode('.', $jwt);

    if (count($partes) !== 3) {
        return false;
    }

    [$headerB64, $payloadB64, $signatureB64] = $partes;

    $firmaEsperada = hash_hmac(
        'sha256',
        "$headerB64.$payloadB64",
        $clave,
        true
    );

    $firmaEsperadaB64 = base64url_encode($firmaEsperada);

    if (!hash_equals($firmaEsperadaB64, $signatureB64)) {
        return false;
    }

    $payload = json_decode(base64url_decode($payloadB64), true);

    if ($payload['exp'] < time()) {
        return false;
    }

    return $payload;
}
```

Imagen de la validación del token

Incluye una función de validación del token que verifica su estructura, la firma, decodifica el payload y comprueba la fecha de expiración, asegurando que solo se acepten tokens válidos.

search.utils.js

El archivo search.utils.js contiene funciones en JavaScript que facilitan la interacción del usuario con la búsqueda de productos, permitiendo generar consultas dinámicas y actualizar la interfaz según los datos disponibles.

redirectToSearchPage()

```
export function redirectToSearchPage() {
    let conditions = "";
    const name = document.querySelector("#tInputSearch").value;
    const category = document.querySelector("#tSelectCategory").value;

    if (name != "") {
        conditions += conditions ? "&" : "?";
        conditions += `name=${encodeURIComponent(name)}`;
    }

    if (category != "all") {
        conditions += conditions ? "&" : "?";
        conditions += `category=${encodeURIComponent(category)}`;
    }

    if (!conditions) {
        window.location = "/public/index.php";
        return;
    }

    const url = `/search/search.php${conditions}`;
    window.location = url;
}
```

Imagen de la redirección a la página de búsqueda

Obtiene los criterios de búsqueda del usuario, construye los parámetros de la URL de una forma segura y redirige a la página de resultados, o a la principal si no existen los criterios de búsqueda.

fillCategorySelect()

```
export async function fillCategorySelect() {
    const service = new productosService();
    let categories = await service.getCategories();

    const nSelect = document.querySelector('#tSelectCategory');

    categories.forEach(category => {
        const nOpt = document.createElement('option');
        nSelect.appendChild(nOpt);
        nOpt.value = category.id;
        nOpt.textContent = category.nombre;
    });
}
```

Imagen de la función rellenar el select de categorías

Carga de manera asíncrona las categorías disponibles mediante un servicio externo y actualiza el <select> de la interfaz, manteniendo la lista sincronizada con la base de datos.

token.utils.js

El archivo token.utils.js incorpora funciones auxiliares en JavaScript dirigidas a la gestión del token JWT en el lado cliente, para controlar la autenticación, la autorización y la interacción de la interfaz según el estado del usuario.

decodeJWT(token)

```
export function decodeJWT(token) {
    if (!token) return null;

    const partes = token.split('.');
    if (partes.length !== 3) return null;

    let payloadB64 = partes[1];

    payloadB64 = payloadB64.replace(/-/g, '+').replace(/_/g, '/');

    const padding = 4 - (payloadB64.length % 4);
    if (padding !== 4) {
        payloadB64 += '='.repeat(padding);
    }

    try {
        const payloadJSON = atob(payloadB64);
        return JSON.parse(payloadJSON);
    } catch (e) {
        throw new Error('Error decodificando token:', e);
    }
}
```

Imagen de la decodificación del token

Decodifica un token JWT, valida su estructura, convierte Base64 URL a JSON y devuelve el contenido del payload, permitiendo acceder de forma segura a la información del usuario.

checkTokenAndChangeLoginButton(token)

```
export async function checkTokenAndChangeLoginButton(token) {
    const loginButton = document.getElementById('tLnkLogin');
    const service = new UsuarioService();

    if (!token) {
        loginButton.textContent = 'Iniciar Sesión';
        return;
    }

    try {
        var userData = await service.getUserDataByToken(token);

        if (userData && userData.nombre) {
            loginButton.textContent = userData.nombre;

            if (userData.rol && userData.rol.toLowerCase() === 'admin') {
                loginButton.href = '/admin/admin.php';
            } else {
                loginButton.href = '/profile/profile.php';
            }
        } else {
            loginButton.textContent = 'Iniciar Sesión';
        }
    } catch (error) {
        loginButton.textContent = 'Iniciar Sesión';
    }
}
```

Imagen de la función del cambio del botón de login

Actualiza el botón de inicio de sesión según el estado del usuario. Redirige a la página de perfil o administración según el rol, o muestra "Iniciar Sesión" si no hay una sesión activa.

isAdmin(token)

```
export async function isAdmin(token) {
    if (!token) return false;

    const service = new UsuarioService();

    try {
        var userData = await service.getUserDataByToken(token);

        if (userData && userData.rol) {
            console.log(userData.rol);
            return userData.rol.toLowerCase() === 'admin';
        } else {
            return false;
        }
    } catch (error) {
        return false;
    }
}
```

Imagen de la función de verificación de administrador

Verifica de manera asíncrona si el usuario tiene los privilegios de administrador, devolviendo un booleano que facilita el control de acceso en la aplicación.

logout()

```
export function logout() {
    localStorage.removeItem('token');
    window.location = "/public/index.php";
}
```

Imagen de la función del cierre de sesión

Cierra la sesión del usuario eliminando el token almacenado y lo redirige a la página principal, esto garantiza la seguridad de la sesión.

Config

Database.php

El archivo Database.php establece una clase Database que gestiona la conexión con la base de datos usando PDO y el patrón Singleton, garantizando que solo exista una conexión activa. El método getInstance() devuelve la conexión, configurada para manejar los errores usando excepciones y devolver los resultados como arrays asociativos. Esto permite usar la misma conexión en todo el proyecto de una forma segura y centralizada.

```php
class Database
{
    private static ?PDO $instance = null;

    private function __construct() {}

    public static function getInstance(): PDO
    {
        if (self::$instance === null) {

            $host = 'mysql';
            $db   = 'tienda_online';
            $user = 'admin';
            $pass = 'admin';

            try {
                self::$instance = new PDO(
                    "mysql:host=$host;dbname=$db;charset=utf8mb4",
                    $user,
                    $pass,
                    [
                        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
                    ]
                );
            } catch (PDOException $e) {
                die("Error en la conexión con la base de datos: " . $e->getMessage());
            }
        }

        return self::$instance;
    }
}
```

Imagen de la clase Database.php

Api

carrito_add.php

El fichero carrito_add.php añade un producto al carrito de compra de un usuario. Para ello, valida la autenticidad del usuario mediante unos tokens JWT, y gestiona la comunicación con la base de datos para mantener actualizada la información del carrito y de los productos.

Uno de los elementos clave del fichero es la validación del usuario, que se realiza a través de un token JWT para asegurar que la operación esté asociada a un usuario autenticado.

```php
$headers = array_change_key_case(getallheaders(), CASE_LOWER);

if (!isset($headers['authorization'])) {
    http_response_code(401);
    echo json_encode([
        'ok' => false,
        'message' => 'Token no proporcionado'
    ], JSON_UNESCAPED_UNICODE);
    exit;
}

$authHeader = $headers['authorization'];
$token = str_replace('Bearer ', '', $authHeader);

$payload = validateJWT($token, $CLAVE_JWT);

if (!$payload) {
    http_response_code(401);
    echo json_encode([
        'ok' => false,
        'message' => 'Token inválido o expirado'
    ], JSON_UNESCAPED_UNICODE);
    exit;
}

$usuarioId = $payload['sub'];
```

Imagen de la validación del usuario autenticado

Además, el sistema verifica la existencia del producto y la disponibilidad de stock antes de añadirlo al carrito, garantizando el orden del inventario y la solidez de los datos.

```php
$stmt = $pdo->prepare("SELECT id, stock FROM productos WHERE id = ?");
$stmt->execute([$productoId]);
$producto = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$producto) {
    http_response_code(404);
    echo json_encode([
        'ok' => false,
        'message' => 'Producto no encontrado'
    ], JSON_UNESCAPED_UNICODE);
    exit;
}

if ($producto['stock'] <= 0) {
    http_response_code(400);
    echo json_encode([
        'ok' => false,
        'message' => 'Producto sin stock disponible'
    ], JSON_UNESCAPED_UNICODE);
    exit;
}
```

Imagen del control de stock del producto

carrito_clear.php

El archivo carrito_clear.php vacía el carrito de la compra de un usuario autenticado. La operación se realiza mediante solicitudes POST y valida la identidad del usuario mediante un token JWT antes de eliminar los productos asociados en la base de datos.

Un elemento que destaca es el uso de una transacción de base de datos al eliminar los productos del carrito. Esto garantiza que si ocurre algún error durante la eliminación, los cambios se revierten automáticamente, asegurando la seguridad de los datos y evitando que el carrito se quede en un estado inconsistente.

```php
try {
    $pdo->beginTransaction();

    $stmt = $pdo->prepare("
        DELETE FROM carrito_productos
        WHERE carrito_id = ?
    ");
    $stmt->execute([$carritoId]);

    $pdo->commit();

    echo json_encode([
        'ok' => true,
        'message' => 'Carrito vaciado correctamente'
    ], JSON_UNESCAPED_UNICODE);

} catch (Exception $e) {
    $pdo->rollBack();
    http_response_code(500);
    echo json_encode([
        'ok' => false,
        'message' => 'Error al vaciar el carrito'
    ], JSON_UNESCAPED_UNICODE);
}
```

Imagen de la transacción del carrito

carrito_delete.php

El archivo carrito_delete.php elimina un producto del carrito de un usuario autenticado. La operación se realiza mediante solicitudes POST y valida la identidad del usuario a través de un token JWT antes de modificar la información del carrito en la base de datos.

Resalta la lógica que controla la eliminación del producto. El sistema primero verifica que el producto esté en el carrito y luego decide si reducir la cantidad en 1 o eliminar la línea completa si sólo quedase una unidad. Esto asegura que el carrito se mantenga consistente y evita eliminar productos incorrectamente.

```php
$stmt = $pdo->prepare("
    SELECT id, cantidad
    FROM carrito_productos
    WHERE carrito_id = ? AND producto_id = ?
");
$stmt->execute([$carritoId, $productoId]);
$linea = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$linea) {
    throw new Exception('El producto no está en el carrito');
}

if ($linea['cantidad'] > 1) {
    $stmt = $pdo->prepare("
        UPDATE carrito_productos
        SET cantidad = cantidad - 1
        WHERE id = ?
    ");
    $stmt->execute([$linea['id']]);
} else {
    $stmt = $pdo->prepare("
        DELETE FROM carrito_productos
        WHERE id = ?
    ");
    $stmt->execute([$linea['id']]);
}
```

Imagen sobre la gestión segura de la cantidad del producto

carrito_get.php

El archivo carrito_get.php obtiene todo el contenido del carrito de un usuario autenticado. La operación se realiza mediante solicitudes POST y valida la identidad del usuario mediante un token JWT antes de recuperar los datos del carrito desde la base de datos.

Sobresale la consulta SQL que une varias tablas para obtener toda la información necesaria de los productos en el carrito. La consulta incluye los datos del producto, como el nombre, el precio, la imagen y la categoría, junto con la cantidad en el carrito, permitiendo al sistema devolver una respuesta completa y estructurada en formato JSON. Esto facilita que el frontend pueda mostrar todos los detalles del carrito.

```php
$carritoId = $carrito['id'];

$stmt = $pdo->prepare("
    SELECT
    cp.id AS lineaId,
    cp.cantidad,
    p.id AS productoId,
    p.nombre,
    p.precio,
    p.imagen,
    c.nombre AS categoria
    FROM carrito_productos cp
    JOIN productos p ON cp.producto_id = p.id
    JOIN categorias c ON p.categoria_id = c.id
    WHERE cp.carrito_id = ?
    ORDER BY cp.id ASC;
");
$stmt->execute([$carritoId]);
$productos = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

Imagen de la consulta que obtiene la información de los productos del carrito

categoria.php

El archivo categoria.php obtiene la información de una categoría específica a partir de su identificador. La operación se realiza mediante GET, validando que el identificador proporcionado sea numérico y existente en la base de datos antes de devolver los datos en formato JSON.

Resalta la consulta que recupera la información de la categoría. La sentencia SQL obtiene el ID, nombre y descripción de la categoría solicitada, asegurando que se obtengan datos válidos y existentes. Esto garantiza la coherencia de la información y le permite al cliente mostrar o procesar correctamente los datos de la categoría.

```php
$id = (int)$_GET['id'];

$stmt = $pdo->prepare("
    SELECT id, nombre, descripcion
    FROM categorias
    WHERE id = ?
");

$stmt->execute([$id]);
$categoria = $stmt->fetch();
```

Imagen sobre la recuperación de los datos de la categoría

categorias.php

El archivo categorias.php obtiene la lista completa de categorías almacenadas en la base de datos. Devuelve los datos en formato JSON y gestiona errores en caso de fallos en la conexión o en la consulta.

Se impone la consulta SQL que recupera todas las categorías disponibles. La sentencia obtiene el ID, nombre y descripción de cada categoría y organiza los resultados en un array asociativo. Esto garantiza que la información devuelta sea completa, estructurada y fácilmente utilizable por el cliente para mostrar o procesar los datos.

```php
$stmt = $pdo->query("
    SELECT
        id,
        nombre,
        descripcion
    FROM categorias
");

$categorias = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

Imagen sobre la recuperación de todas las categorías

login.php

El archivo login.php gestiona la autenticación de usuarios mediante el correo electrónico o el nombre de usuario y la contraseña. Valida los datos que se reciben y, si son correctos, genera un token JWT para permitir el acceso seguro a la aplicación.

La sección más importante es la que verifica la contraseña usando password_verify y genera el token JWT con la información del usuario y la expiración de la sesión. Esto garantiza que solo usuarios autenticados puedan acceder y que las sesiones sean seguras.

```php
if (!$usuario || !password_verify($password, $usuario['password'])) {
    http_response_code(401);
    echo json_encode([
        'ok' => false,
        'message' => 'Usuario o contraseña incorrectos'
    ], JSON_UNESCAPED_UNICODE);
    exit;
}

$clave = 'CLAVE_SECRETA';
$payload = [
    'sub' => $usuario['id'],
    'nombre' => $usuario['nombre'],
    'rol' => $usuario['rol'],
    'iat' => time(),
    'exp' => time() + 3600 * 2
];

$jwt = generateJWT($payload, $clave);

echo json_encode([
    'ok' => true,
    'token' => $jwt
]);
```

Imagen que valida las credenciales del usuario y genera un token seguro

ofertas.php

El archivo ofertas.php obtiene las ofertas activas de los productos de la tienda. Recupera información como el nombre del producto, el precio original y el precio con oferta, y devuelve los resultados en formato JSON para ser consumidos por el cliente.

El elemento más relevante del fichero es la consulta SQL que une las tablas de productos y ofertas para obtener únicamente las promociones activas en el rango de fechas correspondiente. Esto asegura que la información devuelta sea correcta y actual, permitiendo al sistema mostrar al usuario únicamente las ofertas actuales.

```php
$stmt = $pdo->prepare("
    SELECT
        p.id AS producto_id,
        p.nombre AS nombre,
        p.categoria_id,
        p.precio AS precio_original,
        o.precio_oferta AS precio_nuevo,
        o.fecha_inicio,
        o.fecha_fin
    FROM ofertas o
    JOIN productos p ON o.producto_id = p.id
    WHERE o.activa = 1 AND NOW() BETWEEN o.fecha_inicio AND o.fecha_fin
");

$stmt->execute();
$ofertas = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

Imagen que obtiene todas las ofertas activas de productos

pedido_add.php

El archivo pedido_add.php crea un pedido para un usuario autenticado registrando los productos seleccionados. Verifica stock, aplica posibles ofertas y calcula subtotal, IVA, gastos de envío y total, devolviendo un desglose completo en formato JSON.

El fragmento más importante es la verificación de que cada producto exista y tenga suficiente stock antes de añadirlo al pedido. Esta validación previene inconsistencias en el inventario y asegura que solo se registren pedidos válidos.

```php
if (!is_numeric($productoId) || $cantidad <= 0) {
    continue;
}

$stmt = $pdo->prepare("
SELECT
    p.precio AS precio_original,
    p.stock,
    p.nombre,
    o.precio_oferta
FROM productos p
LEFT JOIN ofertas o ON
    o.producto_id = p.id
    AND o.activa = 1
    AND NOW() BETWEEN o.fecha_inicio AND o.fecha_fin
WHERE p.id = ?
");

$stmt->execute([$productoId]);
$producto = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$producto) {
    throw new Exception("El producto $productoId no existe");
}

if ($producto['stock'] < $cantidad) {
    throw new Exception("Stock insuficiente para el producto '{$producto['nombre']}'. Stock disponible:
    {$producto['stock']}, solicitado: {$cantidad}");
}
```

Imagen sobre la validación de stock de un producto

pedidos_get.php

El archivo pedido_get.php obtiene todos los pedidos de un usuario autenticado, incluyendo los productos de cada pedido, la cantidad y el total. Devuelve la información en formato JSON para que el cliente pueda mostrar un historial de pedidos completo.

La sección más relevante es la consulta que recupera los productos asociados a cada pedido. Esta parte une las tablas pedido_productos y productos para obtener el nombre, el precio y la cantidad de cada artículo, asegurando que la información devuelta sea completa y consistente.

```php
$stmt = $pdo->prepare("
    SELECT
        p.nombre,
        pp.precio_unitario AS precio,
        pp.cantidad
    FROM pedido_productos pp
    JOIN productos p ON p.id = pp.producto_id
    WHERE pp.pedido_id = ?
");

$stmt->execute([$pedido['id']]);

$productos = $stmt->fetchAll();
```

Imagen sobre la obtención de productos de un pedido

productos_delete.php

El archivo producto_delete.php permite a un administrador eliminar un producto de la base de datos de la tienda. Verifica que el usuario esté autenticado y tenga rol de administrador antes de proceder a la eliminación, devolviendo un mensaje de confirmación o error en formato JSON.

La sección más importante es la que comprueba que el producto existe y lo elimina de la base de datos dentro de una transacción. Esto asegura que no se eliminen productos inexistentes y mantiene la información en caso de error.

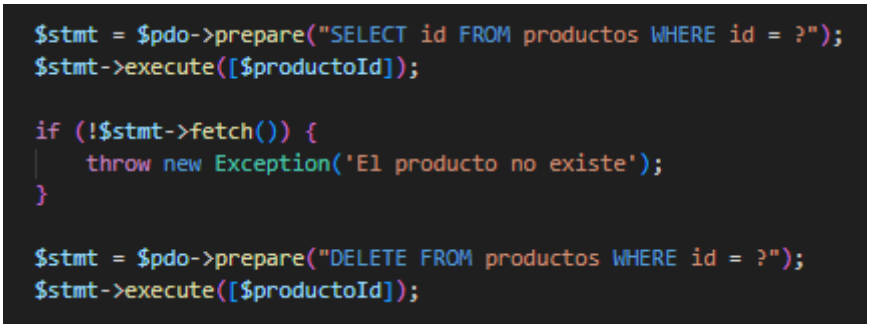

Imagen de la verificación y eliminación de un producto

productos_save.php

El archivo productos_save.php permite a un administrador crear o actualizar productos en la base de datos. Valida los datos recibidos, comprueba la existencia de la categoría y asigna la imagen correspondiente, devolviendo un mensaje de éxito o error en formato JSON.

La sección más importante es la que valida si el producto existe y, según corresponda, realiza un UPDATE o un INSERT en la tabla de productos. Esto asegura que los datos del producto sean correctos y consistentes antes de guardarlos en la base de datos.

```php
if ($productoId) {

    $stmt = $pdo->prepare("SELECT id FROM productos WHERE id = ?");
    $stmt->execute([$productoId]);

    if (!$stmt->fetch()) {
        http_response_code(404);
        echo json_encode([
            'ok' => false,
            'message' => 'Producto no encontrado'
        ]);
        exit;
    }

    $stmt = $pdo->prepare("
        UPDATE productos
        SET nombre = ?, descripcion = ?, precio = ?, stock = ?, categoria_id = ?, imagen = ?
        WHERE id = ?
    ");

    $stmt->execute([
        $nombre,
        $descripcion,
        $precio,
        $stock,
        $categoriaId,
        $imagen,
        $productoId
    ]);

    echo json_encode([
        'ok' => true,
        'message' => 'Producto actualizado correctamente'
    ]);
    exit;
}
```

Imagen sobre la validación y actualización/creación de producto

productos_search.php

El archivo productos_search.php obtiene productos filtrados según nombre o categoría. Valida los parámetros de entrada y devuelve un listado de los productos en formato JSON con la información completa como el nombre, el precio, el stock y la imagen.

La sección más relevante es la construcción dinámica de la consulta SQL según los filtros recibidos. Esto permite aplicar uno o varios criterios de búsqueda de forma flexible y segura, usando prepare para evitar inyecciones SQL y garantizando que solo se devuelvan productos que cumplan los criterios indicados.

```php
$conditions = [];
$params = [];

if (isset($_GET['name']) && trim($_GET['name']) !== '') {
    $conditions[] = 'p.nombre LIKE ?';
    $params[] = '%' . trim($_GET['name']) . '%';
}

if (isset($_GET['category']) && is_numeric($_GET['category'])) {
    $conditions[] = 'p.categoria_id = ?';
    $params[] = (int)$_GET['category'];
}

if (empty($conditions)) {
    http_response_code(400);
    echo json_encode([
        'ok' => false,
        'message' => 'Debes indicar al menos un filtro'
    ], JSON_UNESCAPED_UNICODE);
    exit;
}

$sql = "
    SELECT
        p.id,
        p.nombre,
        p.descripcion,
        p.precio,
        p.stock,
        p.imagen,
        c.id AS categoria_id,
        c.nombre AS categoria
    FROM productos p
    JOIN categorias c ON p.categoria_id = c.id
    WHERE " . implode(' AND ', $conditions);

$stmt = $pdo->prepare($sql);
$stmt->execute($params);
```

Imagen sobre la construcción de la consulta dinámica

productos.php

El archivo productos.php obtiene información de los productos. Si se proporciona un ID, devuelve los datos de un producto específico, sino devuelve todos los productos en la base de datos en formato JSON.

La sección más relevante es la consulta que obtiene un producto concreto por su ID. Se valida que el ID sea numérico y que el producto exista, si no cumple alguna condición se devuelve un error. Esto garantiza que las solicitudes individuales sean seguras y precisas.

```php
$stmt = $pdo->prepare("
    SELECT
        id,
        nombre,
        descripcion,
        precio,
        stock,
        imagen,
        categoria_id
    FROM productos
    WHERE id = ?
");

$stmt->execute([$_GET['id']]);
$producto = $stmt->fetch();

if (!$producto) {
    http_response_code(404);
    echo json_encode([
        'ok' => false,
        'message' => 'Producto no encontrado'
    ], JSON_UNESCAPED_UNICODE);
    exit;
}
```

Imagen sobre la obtención de un producto por ID

usuario.php

El archivo usuario.php obtiene los datos del usuario autenticado mediante un token JWT. Devuelve la información básica como el nombre, el correo electrónico y el rol en formato JSON, asegurando que solo el usuario autorizado pueda acceder a sus datos.

La sección más importante es la que valida el token JWT recibido en los headers y luego obtiene los datos del usuario desde la base de datos. Esto asegura que solo usuarios

autenticados puedan acceder a su información y evita filtraciones de datos de otros usuarios.

```php
$token = str_replace('Bearer ', '', $headers['authorization']);

$payload = validateJWT($token, $CLAVE_JWT);

if (!$payload) {
    http_response_code(401);
    echo json_encode([
        'ok' => false,
        'message' => 'Token inválido o expirado'
    ], JSON_UNESCAPED_UNICODE);
    exit;
}

$usuarioId = $payload['sub'];

$stmt = $pdo->prepare("
    SELECT nombre, email, rol
    FROM usuarios
    WHERE id = ?
");

$stmt->execute([$usuarioId]);

$usuario = $stmt->fetch();
```

Imagen sobre la validación del token y obtención de datos

Frontend

admin.js

En el directorio **admin**, admin.js controla un panel de administración de productos que, tras verificar permisos, permite listar productos y categorías, crear productos nuevos, editar productos existentes, eliminar productos y cerrar sesión.

Se establece y usa la función *setup* que verifica permisos, carga los datos iniciales, pinta la interfaz y conecta los botones con sus acciones.

```
setup();

async function setup() {
    const token = localStorage.getItem("token");

    const admin = await isAdmin(token);

    if (!admin) {
        alert("No tienes permisos para acceder a esta página");
        window.location = "/public/index.php";
        return;
    }

    const service = new productosService();
    const categories = await service.getCategories();
    const products = await service.getProducts();

    await fillCategories(categories);
    await fillTableWithProducts(products, categories);

    const btnAdd = document.querySelector("#tBtnAddProduct");
    btnAdd.addEventListener("click", addProduct);

    const btnLogout = document.querySelector("#tBtnLogout");
    btnLogout.addEventListener("click", logout);
}
```

Función setup de admin.js

El resto de funciones se dividen en dos: las que cambian los datos de los productos, los añaden o borran, llamando a las funciones de la clase *productosService*; y las que hacen el trabajo de pintar la interfaz.

login.js

En el directorio **auth**, login.js controla el inicio de sesión de usuarios en la aplicación y se encarga de validar el formulario de login, autenticar contra el backend y guardar el token. Además, redirige al usuario si el login es correcto y muestra errores si falla

```
function setup() {
    const form = document.querySelector('.loginForm');
    form.addEventListener("submit", validateUser);
}

async function validateUser(e) {
    e.preventDefault();

    const identifier = document.querySelector('#tIdentifier').value;
    const password = document.querySelector('#tPassword').value;

    const authService = new AuthService();
    try {
        const token = await authService.login(identifier, password, 'http://localhost:8081/api/login.php');
        window.localStorage.setItem('token', token);
        window.location.href = '/public/index.php';
    } catch (error) {
        const errorsDiv = document.querySelector('#tDivErrors');
        errorsDiv.textContent = error.message;
        errorsDiv.classList.add('errors');
    }
}
```

Funciones *setup* y *validateUser* de login.js

index.js

En el directorio **public**, index.js carga los productos, muestra ofertas, gestiona el carrusel, filtra por categoría y redirige al detalle del producto.

```
setup();

async function setup() {
    const token = window.localStorage.getItem('token');
    await checkTokenAndChangeLoginButton(token);

    await fillCategorySelect();

    const service = new productosService();
    let productos = await service.getProducts();
    let ofertas = await service.getOffers();

    fillContainers(productos, ofertas);
    fillFeaturedCarousel(productos, ofertas);

    const nSelCategory = document.querySelector('#tSelectCategory');
    nSelCategory.addEventListener('change', async e => {
        await changeTitles(e, service);
        await fillContainersByCategory(e, service);
    });

    const searchBtn = document.querySelector("#tSpnSearch");
    searchBtn.addEventListener("click", redirectToSearchPage);
}
```

Función *setup* de index.js

cart.js

El archivo cart.js gestiona la interacción del usuario con el carrito de compras en el frontend. Obtiene los productos del carrito y las ofertas, actualiza el resumen del pedido y permite añadir o eliminar productos, verificando que el usuario haya iniciado sesión.

La sección más importante es la que obtiene los productos del carrito desde el backend, genera dinámicamente la interfaz con la información de cada producto y actualiza las cantidades y los precios. Esto permite reflejar los cambios en tiempo real y aplicar correctamente los descuentos o las ofertas disponibles.

```
const service = new CarritoService();
const productService = new productosService();

const cartProducts = await service.getCart(token);
const offers = await productService.getOffers();

fillFeaturedContainer(cartProducts, offers);
showNumberOfProducts();
updateOrderSummary();
```

Imagen con la carga de productos y la actualización del carrito

product.js

El archivo products.js se encarga de mostrar la información de un producto en detalle. Obtiene los datos desde el backend, carga la oferta si existe, muestra la imagen, el stock y el precio, y permite añadir el producto al carrito.

La sección clave es donde se traen los datos de los productos, las categorías y las ofertas desde el backend, se obtiene el ID del producto desde la URL, y se llama a la función fillProductWithData para renderizar toda la información. Esto asegura que la interfaz se construya correctamente y que la información del producto se muestre de forma precisa al usuario, incluyendo imágenes, stock y precios.

```
const service = new productosService();
const products = await service.getProducts();
const categories = await service.getCategories();
const offerts = await service.getOffers();

const productId = getIdFromUrl();
if (!productId) return;

fillProductWithData(products, offerts, categories, productId);
const searchBtn = document.querySelector("#tSpnSearch");
searchBtn.addEventListener("click", redirectToSearchPage);
```

Imagen obtención de datos y renderizado del producto

profile.js

El archivo profile.js gestiona la página de perfil del usuario. Obtiene los datos personales y los pedidos desde el backend usando el token JWT, los muestra en la interfaz y permite cerrar sesión. También calcula subtotales, IVA y gastos de envío para cada pedido y alerta si hay productos eliminados de la base de datos.

La parte principal es donde se obtienen los pedidos del usuario y se construyen dinámicamente los elementos HTML para mostrarlos. Cada pedido se representa con un <details> que incluye resumen, lista de productos, subtotales y total con IVA y gastos de envío. Además, se comprueba si algún producto ha sido eliminado para mostrar un aviso, asegurando que el usuario vea información precisa y clara de sus compras.

```
async function loadPedidos(token) {
    try {
        const pedidosService = new PedidosService();
        const pedidos = await pedidosService.getPedidos(token);

        const container = document.getElementById('pedidosContainer');

        if (!pedidos || pedidos.length === 0) {
            container.innerHTML = '<p class="no-pedidos">No tienes pedidos todavía</p>';
            return;
        }

        pedidos.forEach(pedido => {
            const pedidoElement = createPedidoElement(pedido);
            container.appendChild(pedidoElement);
        });

    } catch (error) {
        console.error('Error cargando pedidos:', error);
        document.getElementById('pedidosContainer').innerHTML =
            '<p class="error-pedidos">Error al cargar los pedidos</p>';
    }
}
```

Imagen de la carga y renderizado de pedidos

search.js

El archivo search.js gestiona la página de búsqueda de productos. Obtiene los filtros de nombre y categoría desde la URL, consulta al backend los productos y sus ofertas, y muestra los resultados dinámicamente en las tarjetas clicables, con la imagen, el nombre, la descripción y los precios.

La sección destacable es el renderizado de los productos encontrados. Aquí se crean las tarjetas de cada producto, se asigna la imagen, el nombre, la descripción y los precios (incluyendo ofertas si existen), y se vincula cada tarjeta para que lleve a la página de detalle al hacer clic. Esto asegura que la interfaz sea clara y responsiva para el usuario.

```
products.forEach(product => {
    let productCard = document.createElement("div");
    productCard.classList.add("product-card");
    productCard.setAttribute("data-id", product.id);
    productCard.addEventListener("click", redirectToProductPage);
    productsContainer.appendChild(productCard);

    let productImage = document.createElement("img");
    productImage.setAttribute("src", `/assets/${product.imagen}`);
    productImage.classList.add("product-image");
    productCard.appendChild(productImage);

    let productName = document.createElement("p");
    productName.textContent = product.nombre;
    productName.classList.add("product-name");
    productCard.appendChild(productName);

    let divPrice = document.createElement("div");
    divPrice.classList.add("product-price");
    productCard.appendChild(divPrice);

    const offer = offers.find(offer => offer.producto_id === product.id);

    if (offer) {
        let oldPrice = document.createElement("p");
        oldPrice.textContent = offer.precio_original + " €";
        oldPrice.classList.add("price-old");
        divPrice.appendChild(oldPrice);

        let newPrice = document.createElement("p");
        newPrice.textContent = offer.precio_nuevo + " €";
        newPrice.classList.add("price-new");
        divPrice.appendChild(newPrice);
    } else {
        let price = document.createElement("p");
        price.textContent = product.precio + " €";
        price.classList.add("price-new");
        divPrice.appendChild(price);
    }
    let productDescription = document.createElement("p");
    productDescription.textContent = product.descripcion;
    productDescription.classList.add("product-description");
    productCard.appendChild(productDescription);
});
```

Imagen de DOM que genera cada tarjeta de los productos con/sin ofertas

Services

La carpeta services contiene clases que actúan como intermediarios entre el frontend y la API. Cada servicio maneja las solicitudes HTTP a endpoints específicos para la autenticación, la gestión de productos, el carrito, los pedidos y los datos de usuario, procesando las respuestas y los errores. Esto permite al frontend acceder y manipular datos de manera estructurada y segura sin interactuar directamente con el backend.

auth.service.js

El archivo auth.service.js gestiona la autenticación del usuario. Su función principal es realizar la petición de inicio de sesión al backend, enviando las credenciales y devolviendo el token JWT si la autenticación es correcta.

Centraliza la lógica de login para que otras partes de la aplicación no tengan que gestionar directamente las peticiones fetch ni el tratamiento de errores.

La parte clave del servicio es el método login, donde se envían las credenciales al servidor, se procesa la respuesta y se valida si la operación ha sido correcta. Si el backend devuelve un error, se lanza una excepción; si todo es correcto, se devuelve el token de autenticación.

```
async login(identifier, password, url) {
    let response;
    try {
        response = await fetch(url, {
            method: 'post',
            headers: {
                'content-type': 'application/json',
                'accept': 'application/json'
            },
            body: JSON.stringify({ identifier, password })
        });
    } catch (error) {
        throw new Error(error.message);
    }

    let data;

    try {
        data = await response.json();
    } catch (error) {
        throw new Error(error.message);
    }

    if (!data.ok) {
        throw new Error(data.message);
    }

    return data.token;
}
```

Imagen del proceso de login y validación de respuesta

carrito.service.js

El archivo carrito.service.js gestiona todas las operaciones relacionadas con el carrito de la compra. Se encarga de añadir los productos, eliminarlos, obtener el contenido del carrito y vaciarlo, comunicándose siempre con el backend mediante peticiones protegidas con token JWT.

Este servicio centraliza la lógica del carrito para que la interfaz solo tenga que llamar a métodos claros y reutilizables.

La parte más relevante del fichero es la gestión de las peticiones fetch al backend, incluyendo el envío del token en la cabecera Authorization y el tratamiento de errores tanto de red como de respuesta del servidor. Esto garantiza que el estado del carrito sea coherente y seguro.

```
async addToCart(productID, token) {

    const url = 'http://localhost:8081/api/carrito_add.php';

    let response;
    try {
        response = await fetch(url, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${token}`
            },
            body: JSON.stringify({ productoId: productID })
        });
    } catch (error) {
        alert(`Error de conexión con el servidor: ${error.message}`);
        throw new Error(`Error de conexión: ${error.message}`);
    }

    let data;
    try {
        data = await response.json();
    } catch (error) {
        alert(`Error leyendo la respuesta del servidor: ${error.message}`);
        throw new Error(`Error leyendo la respuesta: ${error.message}`);
    }

    if (!data.ok) {
        alert(data.message || "Error al añadir al carrito.");
        throw new Error(data.message || 'Error al añadir al carrito');
    }

    return data.message;
}
```

Imagen de la comunicación con el backend y el control de errores

endpoints.rest

Este fichero endpoints.rest documenta todos los endpoints de la API disponibles en el backend. Incluye rutas para obtener los productos, las categorías, las ofertas, así como para gestionar el carrito, los pedidos y los datos de usuario. También indica qué método HTTP usar (GET o POST), los headers necesarios (como Authorization con JWT) y la estructura del cuerpo para los POST.

La sección más importante son los endpoints protegidos con JWT, cómo añadir productos al carrito, crear pedidos u obtener información de usuario. Esto asegura que solo los usuarios autenticados puedan realizar acciones sensibles y que la API pueda validar permisos según el rol del usuario.

```
### Añadir producto al carrito
Send Request
POST http://localhost:8081/api/carrito_add.php
Content-Type: application/json
Authorization: Bearer <TOKEN>

{
  "productoId": 2
}
```

Imagen del endpoint que añade un producto al carrito usando un token JWT

pedidos.service.js

El archivo pedidos.service.js se encarga de gestionar todas las operaciones relacionadas con los pedidos del usuario. Permite crear un nuevo pedido a partir de los productos del carrito y obtener el historial de pedidos asociados a un usuario autenticado.

Este servicio actúa como intermediario entre la interfaz y la API de pedidos, simplificando su uso en el frontend.

La parte más importante del fichero es la comunicación segura con el backend usando un token JWT, tanto para crear pedidos como para recuperar el listado. Se valida siempre la respuesta del servidor antes de devolver los datos a la aplicación.

```
async addPedido(productosIds, token) {

    const url = 'http://localhost:8081/api/pedido_add.php';

    let response;
    try {
        response = await fetch(url, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${token}`
            },
            body: JSON.stringify({
                productos: productosIds
            })
        });
    } catch (error) {
        throw new Error(`Error de conexión: ${error.message}`);
    }

    let data;
    try {
        data = await response.json();
    } catch (error) {
        throw new Error(`Error leyendo la respuesta: ${error.message}`);
    }

    if (!data.ok) {
        throw new Error(data.message || 'Error al crear el pedido');
    }

    return data.message;
}
```

Imagen de la creación y la consulta de pedidos protegidos

productos.service.js

El archivo products.service.js centraliza todas las operaciones relacionadas con los productos, las categorías y las ofertas. Se encarga de obtener listados, búsquedas filtradas, detalles de un producto concreto y de realizar acciones de administración como crear, editar o eliminar productos.

Este servicio actúa como una capa de acceso a la API de productos, simplificando la lógica del frontend.

La parte que más destaca del fichero es la gestión de peticiones fetch al backend para recuperar los productos, las categorías y las ofertas, así como la construcción dinámica de los filtros de búsqueda por el nombre y la categoría. Esto permite reutilizar la lógica de acceso a datos en distintas vistas de la aplicación.

```
async searchProducts(name = "", category = "") {
    let conditions = "";

    if (name != "") {
        conditions += conditions ? "&" : "?";
        conditions += `name=${encodeURIComponent(name)}`;
    }

    if (category != "") {
        conditions += conditions ? "&" : "?";
        conditions += `category=${encodeURIComponent(category)}`;
    }

    const url = `http://localhost:8081/api/productos_search.php${conditions}`;
    console.log(url);
    const response = await fetch(url);
    let data;

    try {
        data = await response.json();
    } catch (error) {
        throw new Error(`Error leyendo la respuesta del servidor: ${error.message}`);
    }

    if (!data.ok) {
        throw new Error(data.message || 'Error obteniendo los productos');
    }

    return data.productos;
}
```

Imagen de la obtención y filtrado de productos desde la API

usuario.service.js

El archivo usuario.service.js se encarga de gestionar la información del usuario desde el frontend. Su función principal es obtener los datos de un usuario a partir del token JWT, permitiendo mostrar información personal en la interfaz y controlar el acceso a funcionalidades según el rol del usuario.

Actúa como intermediario entre el frontend y el endpoint /api/usuario.php.

El fragmento más relevante es la llamada a la API con el token JWT, ya que permite autenticar al usuario y obtener su información de manera segura sin exponer las credenciales.

```
export default class UsuarioService {

    async getUserDataByToken(token) {
        const url = 'http://localhost:8081/api/usuario.php';

        let response;
        try {
            response = await fetch(url, {
                method: 'GET',
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            });
        } catch (error) {
            throw new Error(`Error de conexión: ${error.message}`);
        }

        let data;
        try {
            data = await response.json();
        } catch (error) {
            throw new Error(`Error leyendo la respuesta del servidor: ${error.message}`);
        }

        if (!data.ok) {
            throw new Error(data.message || 'Error obteniendo los datos del usuario');
        }

        return data.usuario;
    }
}
```

Imagen de la obtención de datos del usuario desde el token