FICHA TÉCNICA

Raíces

"Conectando tierra y vida"

Desarrollo de Aplicaciones Web

Proyecto de Desarrollo de Aplicaciones Web

Javier Hipólito Rodríguez

Versión 1.0

Nombre del fichero:	DAW_PRW_[RCS]_4. Ficha técnica (4).pdf
Fecha de esta versión:	18-1-26

Historial de revisiones

Fecha	Descripción	Autor
18-1-26	Creación del documento	Javier Hipólito Rodríguez

Índice

1. Introducción

El presente documento técnico describe las tecnologías, herramientas y criterios de diseño utilizados en el desarrollo del proyecto **RAÍCES**, una aplicación web destinada a la gestión de productos locales y pedidos de una cooperativa o pequeños productores.

El proyecto se desarrolla dentro del ciclo formativo de **Desarrollo de Aplicaciones Web (DAW)** y tiene como objetivo aplicar de forma práctica los conocimientos adquiridos en los módulos .

La aplicación ha sido diseñada con un enfoque realista, priorizando:

- La claridad del código.
- El uso de tecnologías estudiadas en el ciclo.
- Una arquitectura mantenible.
- La correcta separación entre frontend, backend y base de datos.
- La posibilidad de ampliación futura.

Este documento recoge de forma detallada todas las decisiones técnicas tomadas y sirve como referencia para la evaluación y defensa del proyecto.

2. Requisitos técnicos

2.1. Plataforma y herramientas de desarrollo software

2.1.1. Requisitos front-end

A continuación, se detallan las tecnologías utilizadas en el desarrollo del frontend, junto con su justificación.

I) Tecnologías / Frameworks

HTML5

Se utiliza como lenguaje de marcado principal para la estructura de todas las páginas de la aplicación.

Permite:

- Crear una estructura semántica.
- Separar contenido y presentación.
- Mantener compatibilidad con cualquier navegador moderno.

Se utiliza principalmente en:

- Páginas públicas (inicio, productos, contacto).
- Estructura base de páginas privadas.

CSS3

CSS se emplea como complemento a Tailwind para:

- Definir variables de color corporativas.
- Ajustes visuales específicos.
- Personalización de ciertos componentes.

Su uso permite mantener flexibilidad sin complicar el diseño.

Tailwind CSS v3.x

Framework principal de estilos del proyecto.

Se utiliza para:

- Diseño responsive.
- Maquetación rápida.
- Tipografías, colores y espaciados.
- Adaptación a dispositivos móviles.

Se ha elegido Tailwind por su eficiencia, limpieza de código y facilidad de mantenimiento, lo que permite centrarse más en la lógica de la aplicación que en el diseño manual.

Flowbite

Flowbite se utiliza como librería de componentes basada en Tailwind CSS.

Se emplea para:

- Modales (login y registro).
- Menús desplegables.
- Barra de navegación responsive.
- Componentes interactivos.

Su uso permite acelerar el desarrollo, garantizar coherencia visual y reducir errores de accesibilidad, ofreciendo una interfaz moderna sin necesidad de crear todos los componentes desde cero.

JavaScript (ES6)

JavaScript se utiliza para toda la lógica del lado cliente.

Funciones principales:

- Comunicación con la API REST mediante fetch().
- Carga dinámica de datos.
- Control del estado de sesión en la interfaz.
- Gestión de eventos y formularios.
- Manipulación del DOM.
- Validaciones básicas.

Se ha optado por JavaScript sin frameworks para mantener el proyecto alineado con los contenidos del ciclo DAW .

II) Otras herramientas de desarrollo

Visual Studio Code
IDE principal utilizado para el desarrollo del proyecto.

Postman
Utilizado para probar y depurar la API REST:

- Envío de peticiones GET y POST.
- Validación de respuestas JSON.
- Comprobación del funcionamiento del backend sin depender del frontend.

Herramientas de desarrollo del navegador (DevTools)
Utilizadas para depurar JavaScript, inspeccionar peticiones de red y analizar errores del frontend.

2.1.2. Requisitos back-end

A continuación se enumeran las tecnologías de desarrollo back-end, versiones y herramientas de apoyo:

I) Plataforma de desarrollo: PHP

PHP 8.0.30 (sin framework)

PHP se utiliza como lenguaje del lado servidor para implementar la lógica de negocio y la API REST.

Funciones principales:

- Autenticación de usuarios (login/logout).
- Gestión de sesiones (me.php).
- Endpoints de productos/pedidos,
- Lógica de roles (admin/cliente),

- Acceso a base de datos.
- Gestión de pedidos y productos.

Se ha optado por PHP porque es la tecnología principal del módulo DWES y es adecuada para una aplicación web con API. Se ha optado por PHP sin framework para mantener el control total del flujo del programa y demostrar lo aprendido del lenguaje, permite demostrar real de sesiones, PDO, rutas y estructura del proyecto. La prioridad es entregar un sistema funcional y defendible.

a) Framework, runtime y versiones

- **PHP 8.0.30**
- **Servidor web Apache 2.4**
- **XAMPP 8.0.30**
- **Server API: Apache 2.0 Handler**

Extensiones utilizadas:

- **PDO**
- **JSON**
- **Sesiones PHP**

Apache + PHP es un entorno estándar, fácil de replicar y muy común en hosting. PDO es clave por seguridad (consultas preparadas) y por buenas prácticas (separación y reutilización).

b) Esquema de desarrollo

Arquitectura **cliente-servidor** con separación:

- **Frontend**: HTML, CSS, JavaScript.
- **Backend**: endpoints PHP (API REST)
- **BBDD**: MySQL

Control de acceso por **sesión** y **roles**.

La aplicación sigue una arquitectura cliente–servidor con separación de responsabilidades, donde el frontend se encarga de la presentación y la interacción con el usuario, mientras que el backend gestiona la lógica de negocio, la autenticación mediante sesiones y el acceso a la base de datos a través de una API REST desarrollada en PHP. Esta separación mejora la seguridad, el mantenimiento y la escalabilidad del sistema.

c) IDE

Visual Studio Code

- Extensiones utilizadas:
- PHP Intelephense

- PHP Debug
- GitLens

d) Otras herramientas o gestores

- **Git** para control de versiones.
- **GitHub** como repositorio remoto.
- **XAMPP** (entorno local: Apache + PHP + MySQL)

e) Otras dependencias

- PDO MySQL (conector base de datos)
- JSON como formato de intercambio
- Control de sesión con $_SESSION
- Fetch API

Son dependencias mínimas pero suficientes. Se prioriza estabilidad y defensa clara frente a añadir capas innecesarias.

2.2. Plataforma de Ejecución

2.2.1. Servidor

Servidor web y versión: **Apache 2.4**
Sistema operativo y versión: **Windows 10/11**
Runtime/Intérprete y versión: **PHP 8.0.30**
Entorno**: XAMPP**

Observaciones a la configuración:

- La aplicación se ejecuta en local mediante XAMPP con configuración estándar.
- Se requiere habilitar extensiones típicas de PHP (PDO, json, session).
- La estructura está preparada para migración a hosting PHP tradicional.

se elige este entorno porque es el que utilizo en el ciclo y el que permite un desarrollo rápido y reproducible. Además, es compatible con un despliegue futuro sin reescribir la lógica.

2.2.2. Alojamiento físico/ Entornos de ejecución

Desarrollo: **Localhost con XAMPP**
Pre-Producción/Pruebas: **No definido (opcional en futuras fases)**
Producción: **Pendiente**

En el alcance actual del proyecto (académico), se trabaja en entorno local. El diseño, no obstante, está pensado para poder desplegarse en un servidor estándar de PHP/MySQL en el futuro.

2.3. Puntos de acceso a la aplicación

2.3 Puntos de acceso a la aplicación

Desarrollo:

- **http://localhost/Raices/**

Pre-Producción:

- No definido

Producción:

- No definida

Mantener rutas claras ayuda a evitar errores de integración y facilita la defensa del flujo real: frontend → API → BBDD.

3. Bases de Datos

Base de datos: **MySQL**
Versión: **8.0.30**
Esquema: **raices**

Justificación de MySQL:

- Es el motor estudiado y utilizado en el ciclo.
- Encaja perfectamente con un modelo relacional (usuarios, productos, pedidos).
- Permite trabajar con integridad referencial y consultas complejas de forma fiable.

Estructura general (tablas principales):

- usuarios: autenticación y roles.
- productos: catálogo, precios, stock, categoría, imagen.
- pedidos: cabecera del pedido (usuario, fecha, estado).

- detalle_pedido: líneas de pedido (producto, cantidad, precio).

Justificación del modelo:
Se separa pedido y detalle para seguir el diseño típico de e-commerce: evita duplicidades y permite historificar pedidos aunque cambie el precio de un producto en el futuro.

4. Interfaces externas

La aplicación incluye una **API REST propia** desarrollada en PHP, consumida desde JavaScript mediante fetch().

- Formato: **JSON**
- Tipo: **REST**
- Comunicación mediante JSON.
- Consumo desde JavaScript mediante fetch().
- Separación clara entre cliente y servidor.
- Posibilidad de ampliación futura (app móvil, frontend React).

La API permite desacoplar frontend y backend, mantener un flujo profesional (cliente consume recursos) y facilita futuras ampliaciones (app móvil o React) sin reescribir el servidor.

5. Seguridad

Durante el desarrollo de la aplicación web *Raíces* se han aplicado diversas medidas de seguridad básicas orientadas a proteger la autenticación de usuarios, el acceso a funcionalidades privadas y la comunicación entre cliente y servidor.

Las siguientes medidas se han implementado o planificado teniendo en cuenta el alcance del proyecto y los contenidos del ciclo formativo.

1) Sesiones PHP para autenticación de usuarios (sesión.php)

La aplicación utiliza sesiones PHP para gestionar el estado de autenticación de los usuarios tras el inicio de sesión.

Una vez que el usuario se autentica correctamente, el backend almacena en la sesión únicamente la información necesaria (identificador, nombre y rol).

Esto permite:

- Mantener la sesión activa entre peticiones.

- Evitar reenviar credenciales en cada solicitud.
- Centralizar el control de acceso en el servidor.

Además, se ha configurado la cookie de sesión con parámetros básicos de seguridad:

```
function start_session(): void
{
   if (session_status() === PHP_SESSION_ACTIVE) {
      return;
   }
   session_set_cookie_params([
      'lifetime' => 0,
      'path' => '/',
      'httponly' => true,   // JavaScript no puede acceder a la cookie
      'secure' => false,
      'samesite' => 'Lax', // ayuda a mitigar ataques CSRF básicos
   ]);

   session_start();
}
```

Justificación:
El uso de httponly evita que JavaScript pueda leer la cookie de sesión, protegiendo frente a ataques XSS que intenten robar la sesión del usuario.

2) Control de roles (administrador / cliente)

La aplicación diferencia entre roles de usuario, permitiendo restringir tanto vistas como endpoints según el tipo de usuario autenticado.

- Los administradores acceden al panel de gestión.
- Los clientes acceden a la tienda y a sus pedidos.

Antes de permitir el acceso a páginas privadas o endpoints sensibles, se comprueba el rol almacenado en la sesión.

```
if (!isset($_SESSION['user']) || $_SESSION['user']['role'] !== 'admin') {
  header('Location: /login.html');
  exit;
}
```

Justificación:
Este control impide que un usuario sin permisos pueda acceder a funcionalidades críticas simplemente escribiendo la URL en el navegador.

3) Consultas preparadas con PDO para prevenir inyección SQL

La conexión a la base de datos se realiza mediante PDO utilizando consultas preparadas, evitando la concatenación directa de datos introducidos por el usuario.

```
$stmt = $pdo->prepare(
  "SELECT id, nombre, rol, password
   FROM usuarios
   WHERE email = :email"
);
$stmt->execute(['email' => $email]);
```

Justificación:
Las consultas preparadas separan la lógica SQL de los datos del usuario, protegiendo frente a ataques de inyección SQL y mejorando la seguridad del sistema.

4) Validación de datos en el backend

Aunque el frontend realiza validaciones visuales, nunca se confía únicamente en el cliente. El backend valida siempre los datos recibidos antes de procesarlos.

Ejemplos:

- Comprobación de campos obligatorios.
- Verificación de formatos.
- Control de métodos HTTP permitidos.

Justificación:
El código JavaScript del cliente puede ser manipulado, por lo que la validación en servidor es imprescindible para garantizar la integridad del sistema.

5) Uso exclusivo de POST en el proceso de login

El endpoint de autenticación (auth.php) solo permite peticiones POST, bloqueando accesos por otros métodos como GET.

```
if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
```

```
    http_response_code(405);
    echo json_encode([
        'ok' => false,
        'error' => 'Método no permitido'
    ], JSON_UNESCAPED_UNICODE);
    exit;
}
```

Justificación:
Esto evita que las credenciales puedan enviarse por URL y asegura que el login solo se realice mediante formularios o peticiones controladas.

6) Gestión segura del cierre de sesión (logout)

En el proceso de cierre de sesión se destruye completamente la sesión activa:

- Se eliminan las variables de sesión.
- Se invalida la sesión actual.

Justificación:
Esto evita que una sesión antigua pueda reutilizarse desde el mismo navegador o dispositivo, aumentando la seguridad del sistema.

7) Evitar exponer información sensible en la API

Los endpoints de la API devuelven únicamente la información estrictamente necesaria para el frontend.

Ejemplo:

- Nunca se envían contraseñas.
- No se exponen datos internos de la base de datos.
- Se controla la información devuelta en me.php.

Justificación:
Reducir la información expuesta minimiza el impacto de posibles ataques y sigue el principio de mínima exposición.

8) Contraseñas con hash en base de datos

Se ha previsto la implementación de hash de contraseñas utilizando funciones nativas de PHP como:

- password_hash()
- password_verify()

Justificación:
El uso de hash garantiza que las contraseñas no se almacenen en texto plano, siendo una medida imprescindible en entornos reales de producción.

9) Estructura public y redirección "puente"

La aplicación separa claramente:

- Recursos públicos (public/)
- Lógica del servidor y API

El acceso a páginas privadas se realiza siempre a través de archivos PHP que validan la sesión antes de mostrar contenido.

Justificación:
Esta estructura evita el acceso directo a recursos sensibles y refuerza el control de acceso desde el servidor.

6. Control de versiones

Se utiliza Git como sistema de control de versiones y GitHub como repositorio remoto del proyecto.

El uso de Git permite mantener un historial completo de cambios, revertir errores de forma controlada y trabajar de manera ordenada durante el desarrollo. Además, facilita el seguimiento de la evolución del proyecto y la organización del trabajo por fases.

Durante el desarrollo se ha trabajado con ramas diferentes, lo que ha permitido:

- Probar distintas estructuras de carpetas.
- Evaluar la organización del proyecto antes de consolidarla.
- Realizar cambios sin afectar a la versión principal.
- Comparar enfoques de desarrollo y seleccionar el más adecuado.

Esta forma de trabajo ha permitido analizar la efectividad de la arquitectura del proyecto y mejorar progresivamente su organización

7. Observaciones

El proyecto se ha diseñado con un enfoque incremental: primero asegurar un núcleo funcional (autenticación, roles, catálogo y pedidos) y posteriormente añadir mejoras.

Se ha priorizado:

- Claridad del código.
- Organización del proyecto.
- Uso de tecnologías estudiadas.
- Posibilidad de ampliación futura.

El sistema queda preparado para futuras mejoras como:

- Despliegue en servidor real.
- Implementación de pasarela de pago.
- Panel de estadísticas.
- Evolución a framework backend o frontend.

Justificación:
Se prioriza que el sistema sea funcional, claro de defender y coherente con el tiempo disponible y el aprendizaje del ciclo.