

Nombre del alumno: Oswaldo Domingo

Ciclo: Desarrollo de Aplicaciones Web (DAW)

Memoria del Proyecto de DAW

IES Abastos. Curso 2024/25. Grupo 2DAW. 29 de Noviembre de 2025

Tutor individual: CARCELLER PEREZ, JOSE MARIA

INDICE

1. [identificación, justificación y objetivos del proyecto](#)
2. [diseño del proyecto](#)
3. [Desarrollo del proyecto](#)
4. [Evaluación y conclusiones finales](#)
5. [Referencias](#)
6. [Anexos](#)

1. Identificación, justificación y objetivos del proyecto

Fase I: Identificación de necesidades del sector productivo y de la organización de la empresa

Actividad de la empresa y ubicación en el sector

Se toma como referencia una **agencia inmobiliaria tipo**, de perfil **familiar o de pequeño equipo**, dedicada a la mediación inmobiliaria (compra-venta y alquiler) en un entorno urbano. Su actividad principal se centra en:

- **Captación de inmuebles** y gestión de propietarios.
- **Publicación y marketing** (web propia, portales inmobiliarios, redes sociales).
- **Gestión de leads** (contactos entrantes, seguimiento y cualificación).
- **Valoración (tasación orientativa)** para acelerar la toma de decisiones y la captación.
- **Gestión documental y cumplimiento** (contratos, RGPD, registros y comunicaciones).

Este perfil de agencia prioriza herramientas **ágiles y rápidas**, orientadas a tareas concretas del día a día, frente a plataformas más complejas con curva de aprendizaje media/alta.

Organigrama de la empresa

En una agencia pequeña la estructura suele ser ligera y enfocada a operaciones. Un organigrama típico incluye:

- **Gerencia/Dirección:** toma de decisiones, estrategia comercial y coordinación.
- **Agentes comerciales:** captación, visitas, negociación y cierre.
- **Administración/BackOffice:** documentación, contratos, archivo y soporte a ventas.
- **Marketing/Comunicación** (interno o externo): web, anuncios, fotografía, RRSS.
- **Colaboradores externos:** asesoría jurídica, gestoría y servicios técnicos (reformas, certificados, etc.).

Este contexto condiciona el software: se valora la **simplicidad**, la **velocidad** y la automatización de tareas repetitivas.

Tendencias del sector inmobiliario (2019-2024)

Entre 2019 y 2024 el sector inmobiliario ha mostrado una combinación de **incremento de precios en zonas consolidadas**, mayor peso del comprador/inversor y un fuerte impulso a la **digitalización** del proceso comercial. En paralelo, muchas agencias han reforzado:

- La captación y publicación online (web propia + portales).
- La rapidez en la respuesta al lead (automatización de correos/avisos).
- El uso de datos para apoyar decisiones (comparables, precios/m² y estimaciones orientativas).

Este contexto justifica la incorporación de herramientas ágiles de gestión y módulos específicos como la **tasación orientativa**, orientados a acelerar la captación y mejorar la calidad del seguimiento comercial.

Convenio colectivo aplicable

La relación laboral del personal técnico contratado se rige por el Convenio Colectivo Estatal de Empresas de consultoría y Estudios de Mercados (Convenio TIC), mientras que los agentes comerciales se acogen al convenio sectorial de mediación inmobiliaria.

Cultura de la empresa: Imagen corporativa

La identidad corporativa en una agencia de este perfil busca transmitir **confianza, cercanía y profesionalidad**. A nivel de producto, esto se traduce en:

- Interfaz clara, textos directos y llamadas a la acción visibles.
- Coherencia visual (tipografía, paleta, componentes reutilizables).
- Contenido enfocado a conversión: propiedades, contacto, valoración y captación.

El objetivo es que la web sea un **canal propio** de adquisición y que el back Office sea intuitivo para perfiles no técnicos.

Sistemas de calidad y seguridad

El sistema aplica buenas prácticas de seguridad web y calidad de código: validación y sanitización de entradas (público y back Office), control de acceso por roles, protección de credenciales mediante .env y tokens CSRF en formularios clave.

1.1. Identificación

Título del proyecto: Plataforma de Gestión Inmobiliaria (portal + backoffice) con herramienta de tasación orientativa.

Autor: Oswaldo Domingo.

Ciclo formativo: Desarrollo de Aplicaciones Web (DAW).

Ámbito: Solución orientada a agencias inmobiliarias pequeñas/medianas; el caso de estudio se enmarca en un entorno local, pero la propuesta es extrapolable.

1.2. Justificación

La motivación del proyecto nace de una necesidad habitual en agencias inmobiliarias de tamaño reducido: disponer de un **canal propio** (web) y un **backoffice** que sea **rápido y fácil de usar**, sin depender al 100% de terceros.

En el sector es frecuente encontrar dos extremos:

- **Herramientas “corporativas” o de grandes operadores**, con reglas y flujos rígidos que limitan la personalización y la evolución del negocio.
- **CRMs muy completos y de coste elevado**, que incorporan módulos avanzados que muchas agencias no utilizan, aumentando la curva de aprendizaje y reduciendo la agilidad operativa.

En una inmobiliaria de perfil familiar o de pocos agentes, la prioridad suele ser la **captación de inmuebles**, la atención rápida al lead y la publicación eficiente, no invertir tiempo en aprender funcionalidades que no aportan valor inmediato.

Por ello, este proyecto propone una plataforma modular centrada en lo esencial:

- Portal público orientado a conversión (captación + contacto).
- Backoffice con gestión básica de inmuebles y usuarios/roles.
- Herramienta de tasación como apoyo a captación y cualificación de leads.

El diseño se ha planteado para evolucionar en un contexto profesional, manteniendo el control tecnológico (arquitectura MVC y buenas prácticas de seguridad).

1.3. Objetivos

Objetivo general: Diseñar e implementar una plataforma web para agencias inmobiliarias que combine un portal público y un backoffice de gestión, incorporando una herramienta de tasación orientativa y medidas de seguridad.

Objetivos específicos:

- Implementar arquitectura **MVC** con separación clara de responsabilidades.
- Gestionar datos en **MySQL/MariaDB** con acceso mediante **PDO** y consultas parametrizadas.
- Desarrollar módulos de **autenticación**, control de sesión y **RBAC** (roles/permisos).
- Incorporar protección **CSRF**, validación/sanitización de entradas y manejo de errores.
- Diseñar una interfaz **responsive** (mobile-first) para una navegación consistente.
-

Implementar una herramienta de tasación orientativa orientada a captación de leads, con posibilidad de evolucionar la fuente de datos a base de datos en futuras iteraciones.

2. Diseño del proyecto

Fase II: Diseño del proyecto

Análisis de la realidad local y oferta empresarial

El proyecto se plantea para una agencia inmobiliaria con presencia local y con necesidad de reforzar su captación a través de un canal propio. En ciudades con alta rotación y competencia, la demanda se concentra en:

- **Visibilidad online** (portal propio + portales inmobiliarios).
- **Rapidez de respuesta** al lead (formularios, contacto y seguimiento).
- **Gestión eficiente** del catálogo y su publicación.

Aunque el caso de uso se contextualiza en un entorno urbano, el planteamiento es extrapolable a cualquier agencia que quiera reducir dependencia de terceros y medir mejor su conversión (captación de inmuebles, solicitudes de información y tasaciones orientativas).

Viabilidad técnica y oportunidad del proyecto

La viabilidad técnica es alta al usar tecnologías estándar en desarrollo web (PHP, MySQL/MariaDB, HTML/CSS/JS) y un enfoque MVC sin dependencias pesadas.

La oportunidad del proyecto está alineada con necesidades reales del sector: muchas agencias pequeñas se encuentran entre herramientas rígidas (poco personalizables) y CRMs complejos de coste elevado que no se aprovechan al completo. En este contexto, una solución **modular** y centrada en lo esencial (captación, catálogo, tasación y comunicación) aporta:

- **Agilidad operativa** y menor curva de aprendizaje.
- **Coste controlado** y reducción de dependencia de plataformas externas.
- **Evolución por fases**, añadiendo funcionalidades cuando el negocio lo requiera.

Estructura general y fases del proyecto

El proyecto se ha estructurado en las siguientes fases cronológicas: Fase de análisis (Semanas 1-2): definición de requisitos y diseño de la base de datos. Fase de Desarrollo del Núcleo (Semanas 3-5): implementación del Router, Autoloader y sistema de seguridad. Fase de Desarrollo de Módulos (Semanas 6-10): Construcción del CRUD de usuarios, clientes, inmuebles y demandas. Fase de Pruebas y QA (Semanas 11-12): Validación de seguridad y corrección de errores (debug).

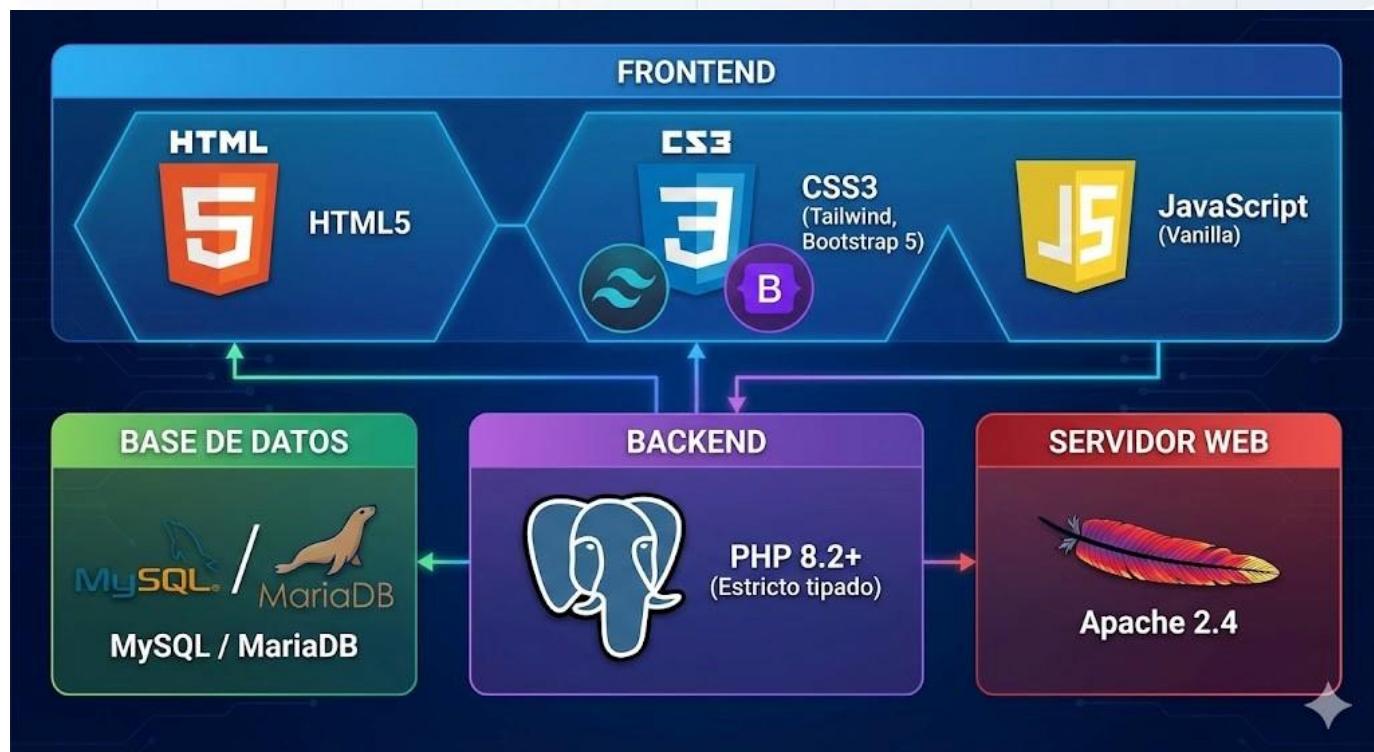
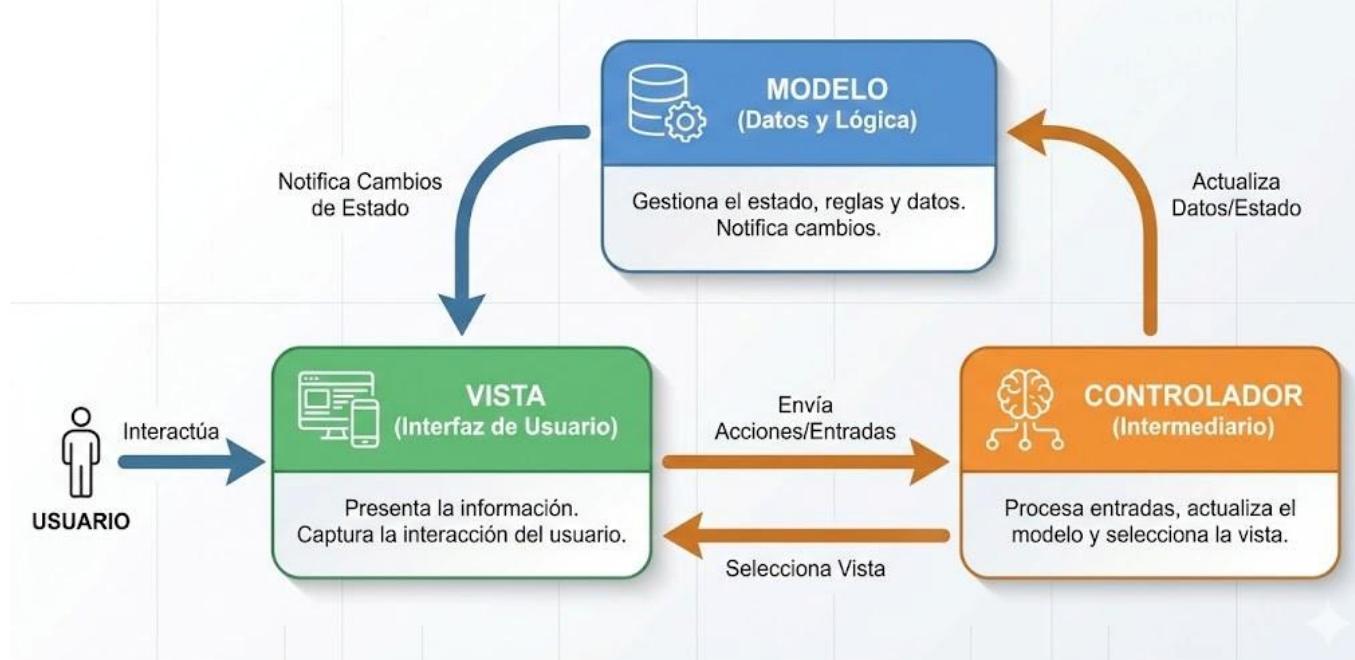
Recursos y necesidades de financiación

Recursos Personales:

Un desarrollador Full Stack (autor del proyecto) y el asesoramiento del tutor individual. Recursos Materiales: Equipo informático (PC de desarrollo), servidor de hosting (Apache/PHP/MySQL), licencias de software (IDE, Git). Viabilidad Económica: El proyecto requiere una inversión inicial mínima (hosting y dominio ~100€/año) y es altamente rentable al automatizar procesos que antes requerían horas de personal administrativo.

2.1 Arquitectura del Sistema

DIAGRAMA ARQUITECTURA MVC (Modelo-Vista-Controlador)



Se ha optado por una arquitectura Modelo-Vista-Controlador (MVC) personalizada, favoreciendo el control total sobre el flujo de la aplicación y el rendimiento. Frontend: HTML5, CSS3 (con Tailwind CSS y Bootstrap 5), JavaScript (Vanilla). Backend: PHP 8.2+ (Estricto tipado). Base de Datos: MySQL / MariaDB. Servidor Web: Apache

2.2. Estructura de Directorios

El proyecto sigue una estructura segura donde solo el directorio `public/` es accesible desde la web (DocumentRoot apuntando a `public/`):

```
/  
+-- app/           # Nucleo de la aplicacion  
|   +-- Controllers/    # Logica de control  
|   +-- Models/        # Acceso a datos  
|   +-- Views/         # Plantillas HTML  
|   +-- Core/          # Router, Database, Config  
+-- config/        # Configuracion del entorno  
+-- database/       # Migraciones y SQL  
+-- docs/           # Documentacion  
+-- logs/           # Logs de aplicacion  
+-- public/          # Entry point (index.php) y assets  
+-- storage/         # Archivos generados por la app  
+-- tests/           # Pruebas  
+-- .htaccess        # Reglas de seguridad y rewrite  
+-- index.php        # Entrada alternativa en raiz
```

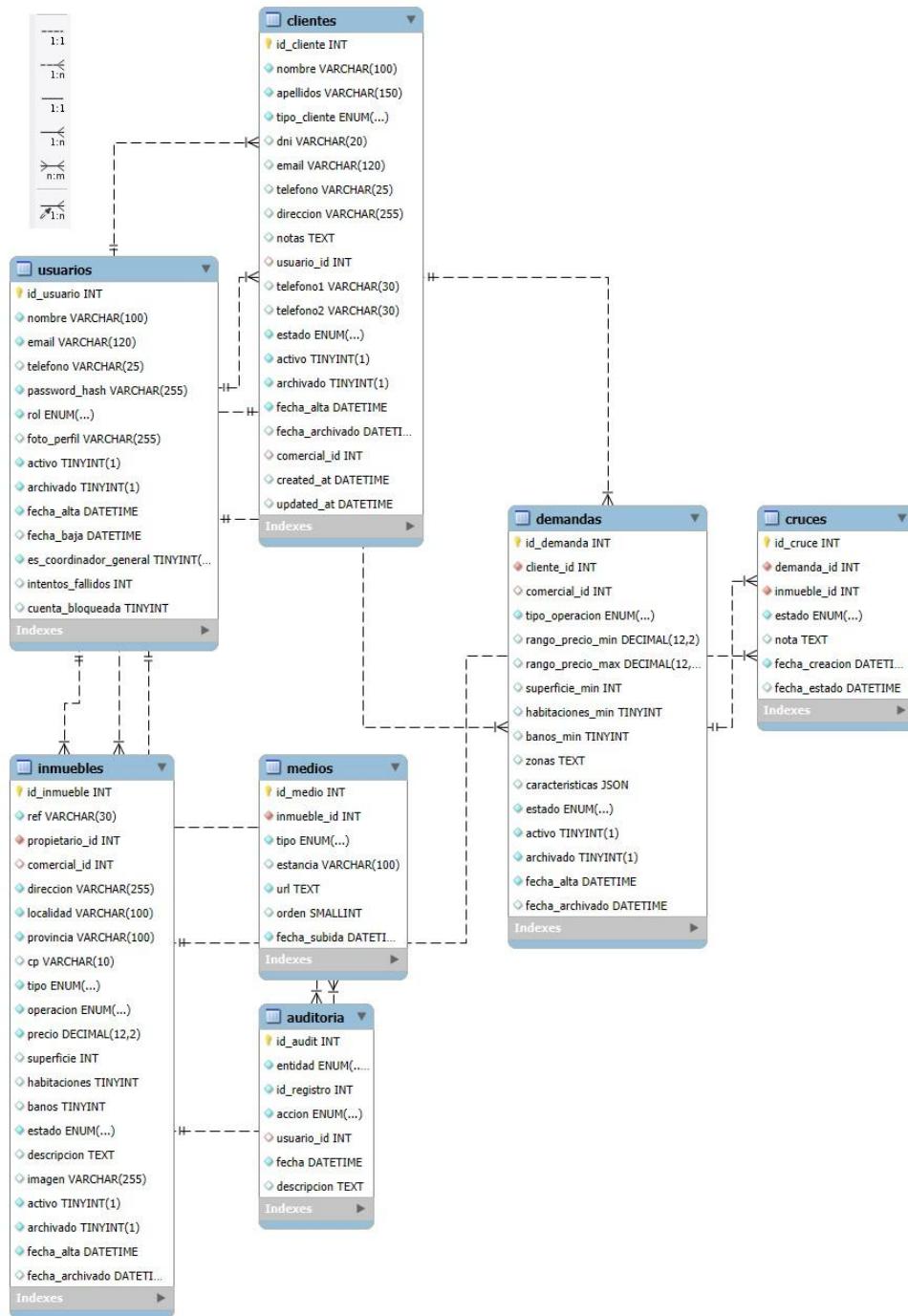
2.3. Diseño de Base de Datos

El esquema relacional se ha diseñado para garantizar la integridad de los datos. Tabla usuarios:

Gestión de acceso y roles. Incluye campos de auditoria (`created_at`, `updated_at`) y seguridad

(`intentos_fallidos`, `cuenta_bloqueada`). Soft

Deletes: Se implementa el borrado lógico mediante columnas `activo` y `archivado` para preservar el histórico de datos.



- **Diagrama del modelo** ve [Anexo B — Diagrama](#) [Diagrama de la base de Notas:](#)

- **Índices recomendados (rendimiento):** clientes.dni , clientes.email , inmuebles.ref , inmuebles.localidad , inmuebles.operacion , demandas.tipo_operacion .
- **Catálogos vs ENUM:** si va a haber una evolución frecuente de valores, puede ser útil pasar tipo , estado y operación a tablas de catálogo.
- **Reglas de borrado en FKs:** documentar en el diagrama si son ON DELETE CASCADE O ON DELETE SET NULL en tablas como auditoria y cruces .
- **Normalización de dirección (opcional):** separar calle/numero/piso solo si se necesitan filtros o informes detallados.
- **Roles en tabla:** alternativa a ENUM si se quiere gestionar roles de forma dinámica.

2.4. Prototipado UI/UX (Figma)

Durante la fase de diseño se elaboró un prototipo en Figma para definir la estructura visual del portal, validar la jerarquía de información y asegurar un comportamiento responsive antes de cerrar la maquetación final.

El diseño se trabajó con enfoque **mobile-first** y se iteró hasta obtener un resultado consistente en los tres formatos principales. Para la documentación se incluye una **captura compuesta** que resume las vistas (móvil, menú móvil y tablet/escritorio).

A continuación, se muestra una captura del **resultado final implementado**:

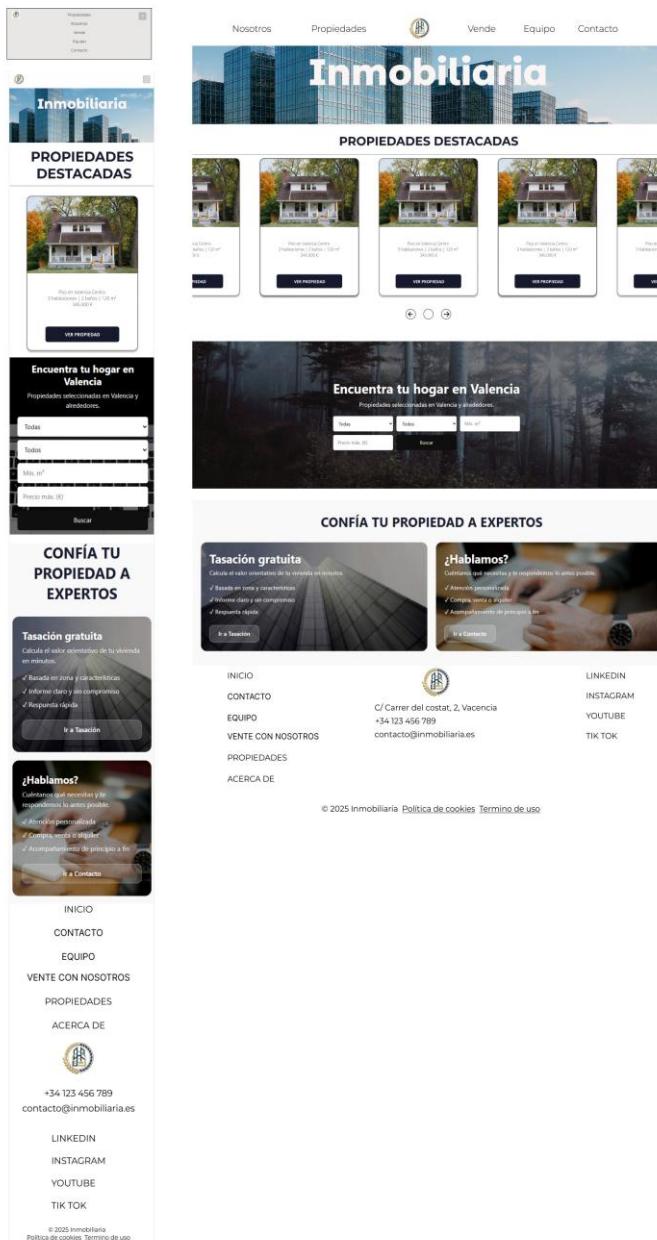


Figura. Landing Page responsive (captura compuesta): vista móvil, menú desplegable y vista tablet/escritorio.

El prototipo se incluye como material complementario en el [Anexo A](#) mediante enlace en modo solo lectura.

3. Desarrollo del proyecto

3.1. Secuenciación y procedimientos de trabajo

El desarrollo se ha realizado siguiendo una metodología ágil e iterativa. Cada módulo (Usuarios, Clientes, Inmuebles, Demandas, etc.) ha seguido el mismo ciclo:

1. **Definición del esquema de datos** (migraciones, relaciones e índices).
2. **Implementación del modelo** y reglas de negocio (validaciones y restricciones).
3. **Desarrollo de controladores y rutas** (Router → Controller@action).
4. **Maquetación de vistas** con Bootstrap y parciales reutilizables.
5. **Pruebas manuales de flujo completo** (alta, edición, permisos por rol, errores controlados).

Control de versiones (Git/GitHub) y trazabilidad

- El proyecto se mantiene en un **repositorio Git** con remoto en [GitHub](#) como *referencia única* del código.
 - Se realizan **commits atómicos** por cambio funcional (core, módulos, UI, documentación),
 - facilitando trazabilidad y revisión.
- Los ficheros sensibles o dependientes del entorno (`.env` , logs y subidas) se excluyen con `.gitignore` y se configuran directamente en cada servidor.

Despliegue y respaldo (Sered + servidor doméstico)

- **Producción (Sered / cPanel):** despliegue sincronizado vía **Git Version Control** (pull desde GitHub) y `public/` como *DocumentRoot*.
- **Servidor doméstico de respaldo (Debian 13):** despliegue del mismo repositorio como entorno espejo para validación y continuidad.
- **Acceso externo (servidor doméstico):** publicación mediante **Cloudflare Tunnel** usando el dominio `oswaldo.dev` y el subdominio `inmobiliaria.oswaldo.dev` .
- **Acceso privado a administración técnica:** uso de **Tailscale** para acceder a servicios sensibles (p. ej.
- **phpMyAdmin**) sin exponerlos públicamente.

Estrategia de actualización: primero se valida en el servidor doméstico; si todo es correcto, se actualiza producción en Sered.

3.2. Plan de prevención de riesgos (PRL)

Dado que el proyecto se desarrolla en un entorno de oficina y teletrabajo, se han identificado y prevenido los siguientes riesgos:

Riesgos ergonómicos: fatiga visual, molestias de espalda por postura.

Medidas: mobiliario ergonómico, pausas periódicas, control de iluminación.

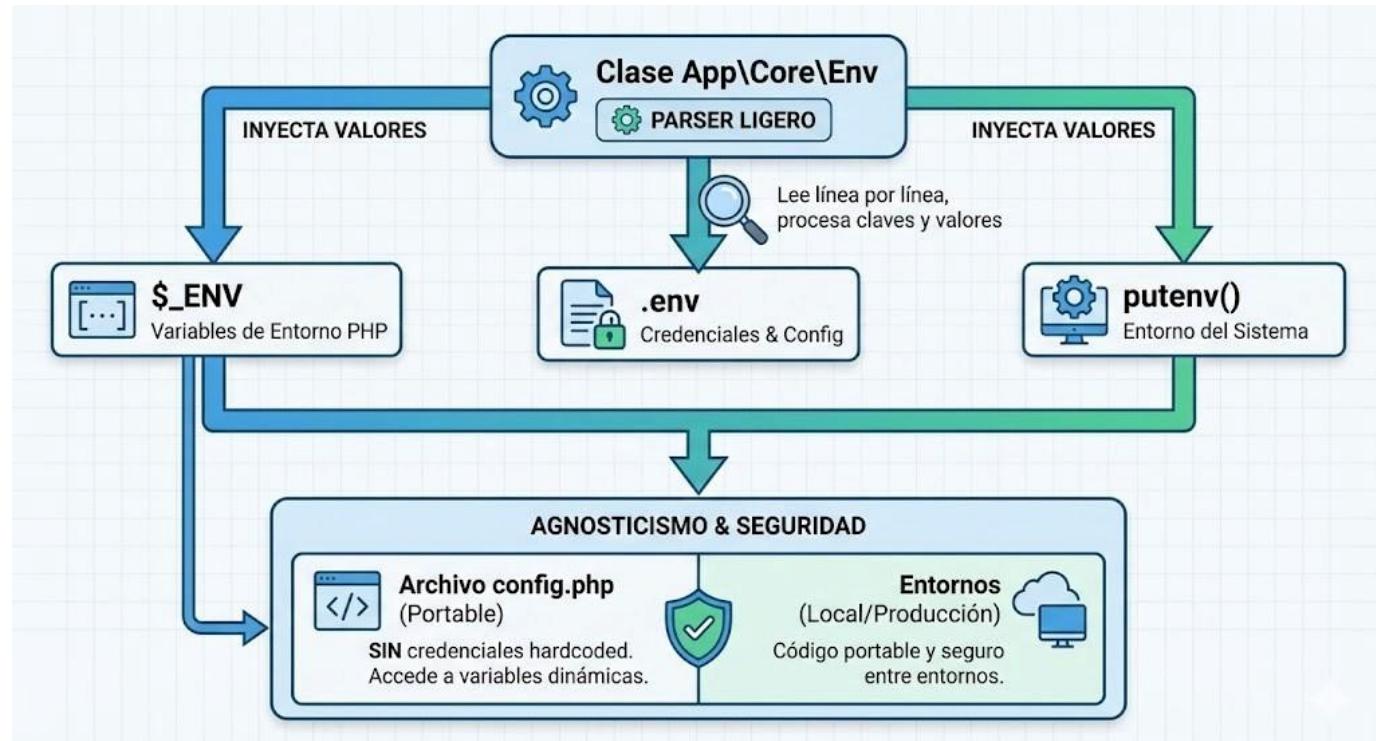
- **Riesgos psicosociales:** estrés por plazos de entrega.
 - Medidas: planificación realista, división por hitos, desconexión digital.
- **Seguridad eléctrica:** riesgo de cortocircuito o sobrecarga en equipos.
 - Medidas: revisión de cableado, regletas con protección.

3.3. Recursos y logística

Se validó la disponibilidad de los recursos necesarios para el desarrollo, despliegue y operación:

- Infraestructura de servidor: entorno local, **producción en Sered (cPanel)** y **servidor doméstico Debian 13** como respaldo.
 - Publicación del servidor doméstico mediante **Cloudflare Tunnel** (sin necesidad de abrir puertos directamente).
 - Administración privada de servicios sensibles mediante **Tailscale** (p. ej. acceso a **phpMyAdmin**), reduciendo superficie de ataque.
 - Librerías externas justificadas (p. ej. **PHPMailer**, licencia LGPL).
- Separación de credenciales mediante .env para portabilidad y seguridad.

3.4. Configuración del entorno



Se implementó un sistema de carga de variables de entorno (.env) para separar la configuración sensible (credenciales de BD, SMTP) del código fuente. Dado que no se utilizan frameworks ni gestores de dependencias como Composer, se desarrolló una solución propia y nativa.

3.4.1. Clase App\Core\Env

- **Parser ligero:** clase propia que lee .env línea por línea, ignora comentarios/líneas vacías y convierte CLAVE=VALOR en variables.
- **Inyección en entorno:** carga en \$_ENV y también en el entorno del proceso con putenv() (accesible con getenv()), mejorando compatibilidad.
- **Agnosticismo:** config.php no contiene credenciales hardcoded; consume solo variables del entorno. El mismo código funciona en local/producción cambiando únicamente el .env .
- **Seguridad:** evita credenciales en repositorios y despliegues. Si faltan variables críticas, el sistema puede fallar de forma controlada.

3.5. Núcleo (Core)

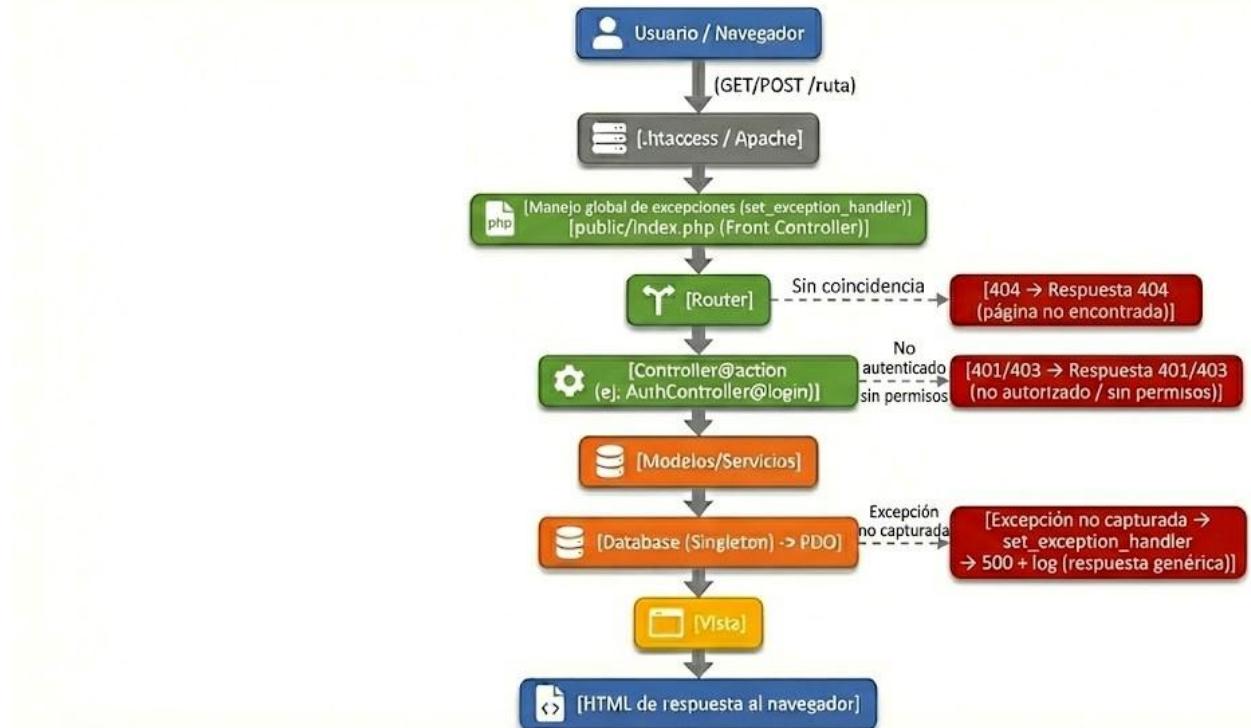


Figura. Flujo de una petición en el núcleo de la aplicación. Todas las peticiones entran por /public/index.php (Front Controller), el Router (/app/Core/Router.php) resuelve la ruta y despacha al controlador correspondiente. Los controladores consumen Modelos/Servicios y acceden a la base de datos mediante una conexión PDO única gestionada por /app/Core/Database.php . El sistema contempla rutas inexistentes (404), accesos no autorizados o sin permisos (401/403) y excepciones de base de datos (500 con registro en logs).

Leyenda de respuestas (HTTP)

- **401 Unauthorized:** el usuario no está autenticado (no hay sesión válida). Normalmente se redirige al login.
 - **403 Forbidden:** el usuario está autenticado, pero no tiene permisos (rol insuficiente).
 - **404 Not Found:** ruta inexistente o no registrada en el Router.
- 500 Internal Server Error:** fallo interno (excepción no controlada); se registra en logs y se devuelve un mensaje genérico.

3.5.1. Front Controller (bootstrap)

El **Front Controller** centraliza el arranque de la aplicación y actúa como *bootstrap* (punto único de entrada). Todas las peticiones llegan a `public/index.php` (vía reglas de rewrite) y desde ahí se inicializa el entorno y se despacha la ruta.

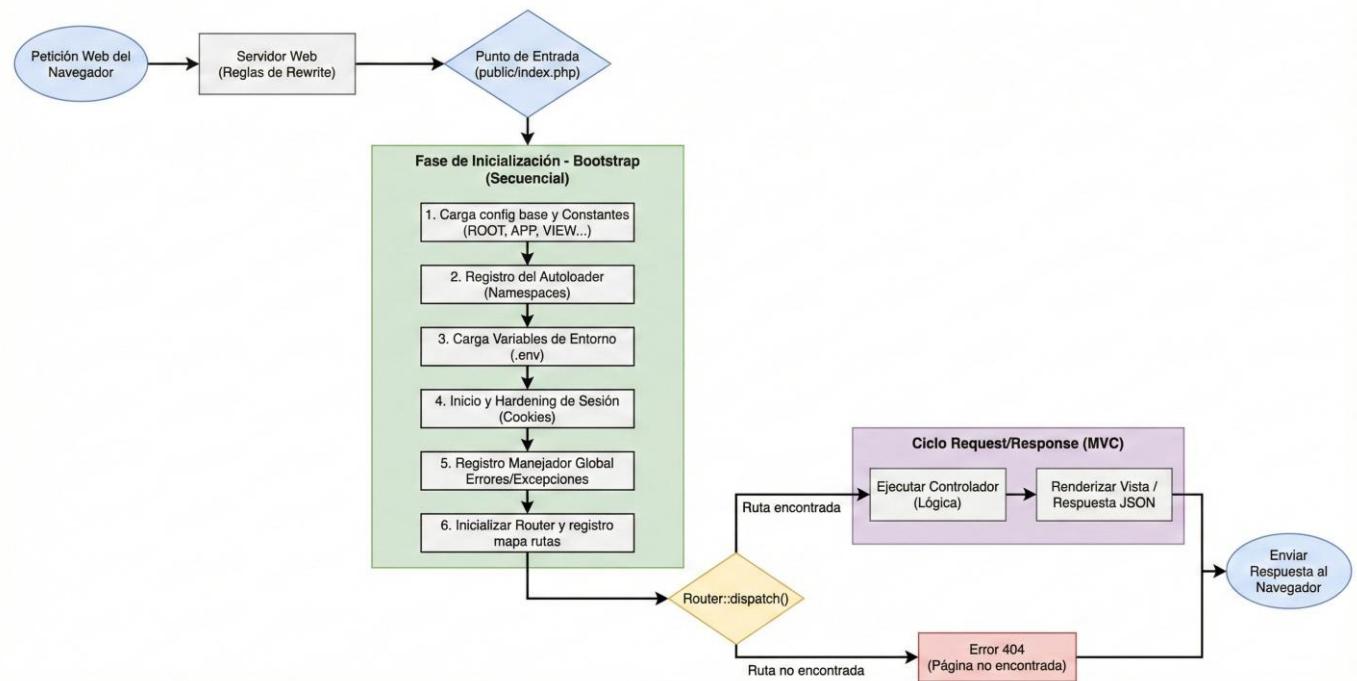
Qué inicializa (bootstrap):

- Carga de configuración base y constantes de rutas (por ejemplo: `ROOT` , `APP` , `VIEW` , etc.).
- Registro del **Autoloader** (carga automática de clases por namespace, sin `require` manuales).
- Carga de variables de entorno con `Env` (`.env`) para credenciales y configuración sensible.
- Inicio y hardening de **sesión** (cookies seguras / regeneración tras login, según el flujo).
- Registro del **manejador global de errores/excepciones** (`set_exception_handler`) para evitar fallos sin control.
- Inicialización del **Router** y registro del mapa de rutas.

Ejecución del ciclo request/response: `dispatch()` → controlador → vista / respuesta.

Ruta del archivo: `/public/index.php`

Flujo de Ejecución Front Controller y Bootstrap (Paso a Paso)



3.5.2. Autoloader

Como el proyecto no utiliza Composer, se implementó un **Autoloader propio** (estilo PSR-4) para cargar clases automáticamente a partir de su namespace.

Qué hace:

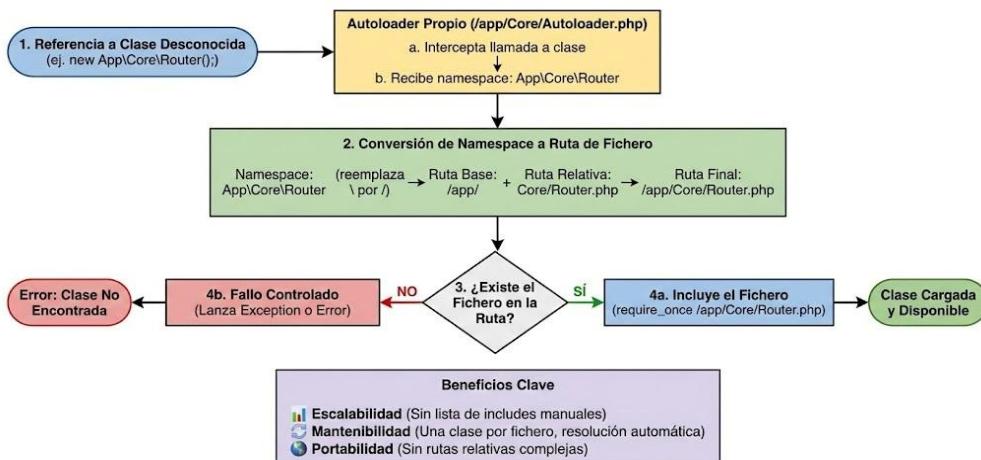
- Intercepta la primera vez que se referencia una clase (sin necesidad de `require` manuales).
- Convierte el namespace (p. ej. `App\Core\Router`) en una ruta de fichero dentro de `/app/` .
- Incluye el fichero si existe; si no, falla de forma controlada (mejor que "silencios" difíciles de depurar).

Por qué es importante:

- Escalabilidad: añadir clases/módulos no obliga a mantener una lista de includes.
- Mantenibilidad: cada clase vive en su archivo y el sistema la resuelve automáticamente.
- Portabilidad: reduce problemas de rutas relativas y acoplamientos.

Ruta del archivo: /app/Core/Autoloader.php (*registrado desde /public/index.php*)

Funcionamiento del Autoloader Propio (Estilo PSR-4)



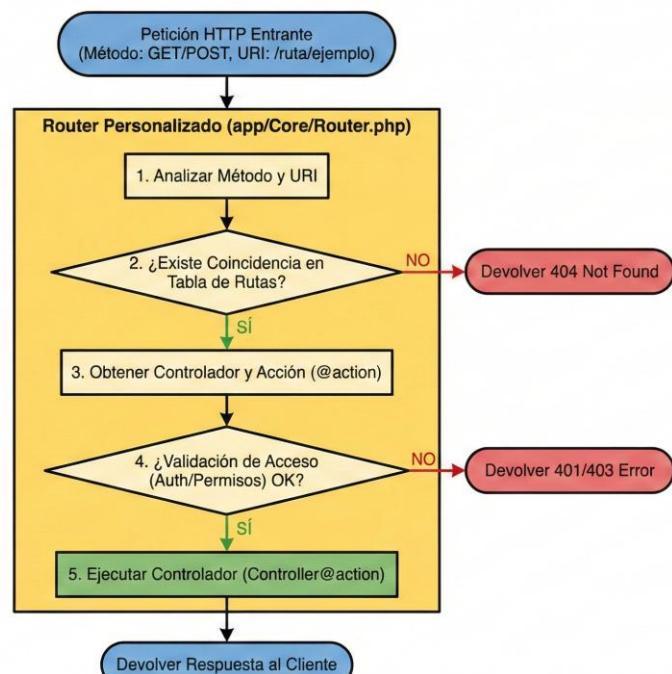
3.5.3. Router

Se desarrolló un enrutador personalizado como punto central de despacho:

- Analiza el método HTTP (GET/POST) y la URI.
- Busca coincidencia en la tabla de rutas.
- Ejecuta el controlador correspondiente (Controller@action) definido en el mapa de rutas.
- Si no existe ruta registrada, devuelve **404 Not Found**.
- Contempla validaciones de acceso (auth/permisos) desde controladores, devolviendo **401/403** cuando corresponda.

Ruta del archivo: /app/Core/Router.php

Flujo de Procesamiento del Router Personalizado (/app/Core/Router.php)



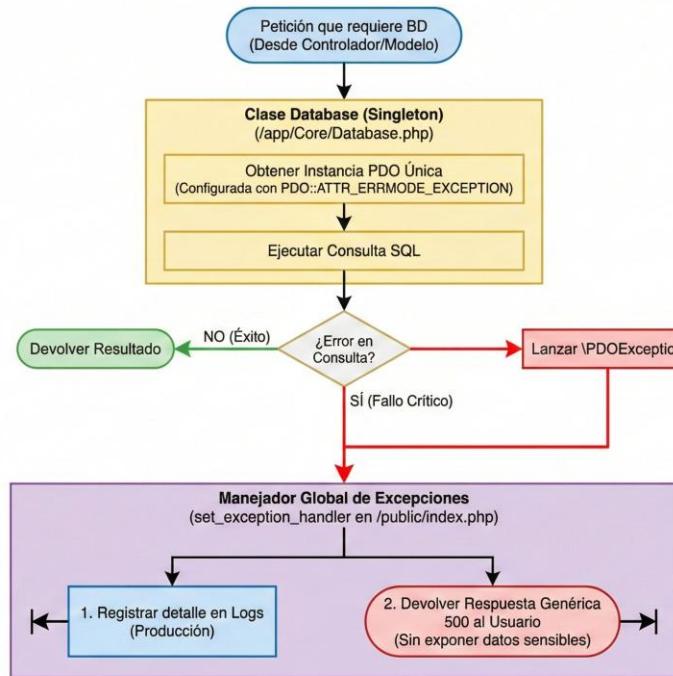
3.5.4. Database

Se implementó una clase **Singleton** para gestionar una única conexión PDO por petición, evitando conexiones repetidas y centralizando configuración:

- PDO configurado para lanzar excepciones (`\PDOException`) ante errores.
- Ante fallo crítico, la excepción se gestiona de forma centralizada desde el Front Controller (`/public/index.php`) mediante `set_exception_handler`.
- En producción se registra el detalle en logs y se devuelve una respuesta genérica **500** al usuario.

Ruta del archivo: /app/Core/Database.php

Flujo de Gestión de Base de Datos y Errores (Singleton PDO)



Relación con el manejo global de errores (ver 3.7): El Core está diseñado para que cualquier error no controlado (por ejemplo, una `\PDOException` en una consulta) no “rompa” la aplicación en pantalla. En su lugar, la excepción se propaga hasta el **handler global** registrado en `/public/index.php`, donde se registra en logs y se responde con una salida controlada (vista de error / respuesta 500) sin exponer información sensible.

3.6. Módulos implementados

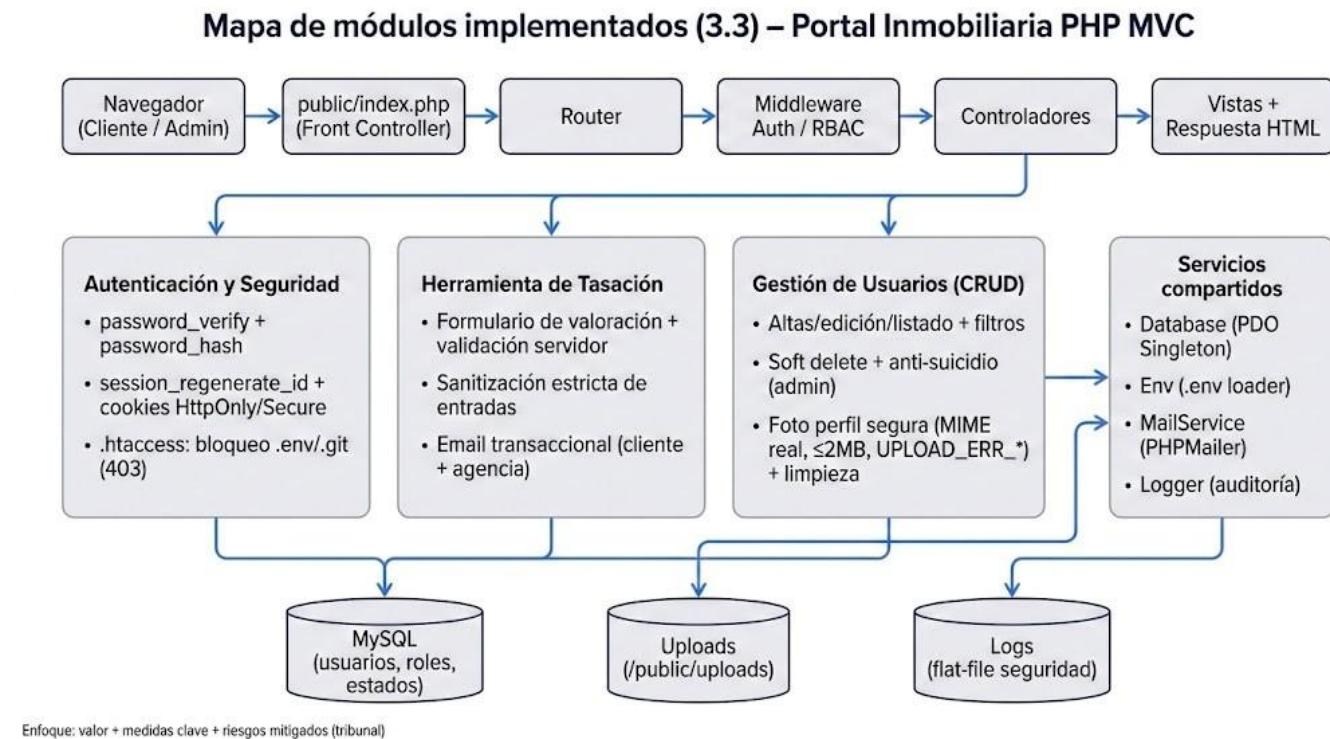


Figura. Mapa de módulos implementados y flujo general del sistema (detalle funcional en el **Anexo C**).

3.6.1. Autenticación y seguridad

Qué aporta al sistema (valor)

- Acceso seguro al panel de administración con contraseñas protegidas mediante hash.
- Sesiones robustas para evitar secuestro/fijación de sesión tras el login.
- Acceso restringido por rol (RBAC) para separar claramente permisos y funcionalidades.

Medidas clave

- **Contraseñas:** password_hash + verificación con password_verify .
- **Sesión:** session_regenerate_id(true) tras login; cookies con **HttpOnly** y **Secure** (en HTTPS).
- **RBAC:** comprobación de autenticación/rol en controladores.
- **Hardening servidor:** reglas .htaccess para bloquear ficheros sensibles (.env , .git , etc.) devolviendo **403**.

Nota de despliegue (seguridad)

- En entorno local existía un enlace visible a "Acceso a profesionales" (`inmobiliaria.loc/login`) para pruebas.

En producción **no se plantea un registro público de usuarios**: el acceso será **interno** (uso de la agencia), por lo que se elimina el enlace desde la zona pública para reducir exposición.

Riesgos mitigados

- Robo de sesión (*session fixation*), accesos no autorizados, exposición de credenciales/configuración.

3.6.2. Herramienta de tasación online

Qué aporta al sistema (valor)

- Captación de leads mediante un formulario de valoración online (flujo directo y medible).
- Automatización de comunicaciones: confirmación al cliente + aviso a la agencia.
- Base reutilizable para otros formularios (plantillas y servicio de correo).

Origen y decisión técnica (Google Sheets como fuente de datos)

- El precio base por zona y reglas/modificadores se obtienen desde una **hoja de Google Sheets** publicada en formato **CSV**.
- Esta decisión surge de un contexto real en el que se buscaba **evitar dependencias del servidor** y permitir que perfiles no técnicos actualizasen valores de forma rápida.
- En el proyecto final se reutiliza el enfoque por su **portabilidad** (cambiar datos sin redeploy) y porque demuestra integración real con herramientas de oficina.

Funcionalidad

- **Formulario público:** /tasacion (datos del inmueble + datos de contacto).
- **Lectura de datos externos:** el front-end descarga y parsea CSV de Google Sheets (p. ej. precios/m² por CP/zona y reglas de ajuste).
- **Algoritmo de valoración:** calcula un rango estimado aplicando ajustes porcentuales sobre el precio base.
- **Comunicación:** envío de dos emails mediante SMTP seguro con **PHPMailer**:
 - Cliente: estimación + confirmación.
 - Agencia: lead con datos para seguimiento.

Seguridad aplicada

- Token **CSRF** en el formulario y validación en el envío.
- Validación y sanitización de entrada en servidor (campos obligatorios y formatos).
- Plantillas HTML separadas para correos corporativos; credenciales SMTP desde .env .

Diagrama de Secuencia: Flujo de Tasación

Resumen: Usuario solicita tasación, el front-end calcula un rango preliminar con datos de Google Sheets, el usuario envía sus datos, el backend valida y envía emails, y finalmente muestra el resultado.

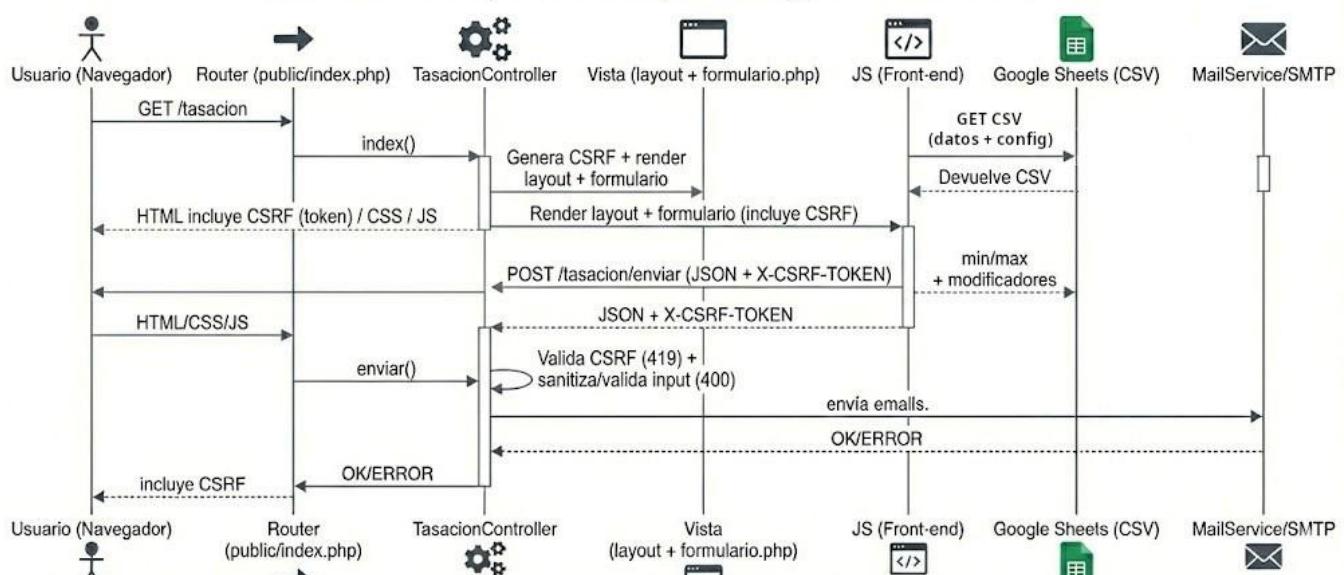


Figura. Diagrama de secuencia del flujo de tasación: el usuario solicita la página, se genera CSRF y se renderiza el formulario; el front-end descarga datos CSV desde Google Sheets para calcular un rango preliminar; el usuario envía los datos (con CSRF), el backend valida y sanitiza, y finalmente se envían correos transaccionales y se muestra el resultado.

Riesgos mitigados

- Inyección de contenido en emails, datos inválidos, exposición de credenciales SMTP.

3.6.3. Gestión de usuarios (CRUD)

Qué aporta al sistema (valor)

- Administración completa de usuarios internos: alta, edición, listado y desactivación.
- Conservación de histórico mediante **baja lógica** (*soft delete*).
- Mejora de UX: usuario identificado en UI (email + foto).

Medidas clave

- **Soft delete** y restricciones por rol.
- **Anti-suicidio:** un admin no puede desactivarse/bloquearse a sí mismo.
- **Subida segura de foto de perfil:**
 - Validación de **MIME real**, control de tamaño (\leq 2 MB) y manejo de UPLOAD_ERR_* .
 - Renombrado aleatorio/único y limpieza de archivos antiguos.
 - Mensajes amigables en UI (sin exponer rutas internas).
- **Auditoría (flat-file):** logs de accesos, fallos y bloqueos con visor integrado.

Riesgos mitigados

- Pérdida de acceso por error humano, subida de archivos maliciosos, falta de trazabilidad ante incidentes.

3.6.4. Gestión de clientes

Qué aporta al sistema (valor)

- Centraliza la gestión de clientes del CRM con **visibilidad por rol** y trazabilidad de asignación
 - comercial.
- Evita incoherencias en datos (DNI duplicado) y protege integridad cuando existen inmuebles asociados.

Migración y esquema

- Migración principal:
 - database/migrations/03_create_crm_tables.sql Tablas:
 - clientes e inmuebles .
 - Relaciones: FKs hacia usuarios y clientes .

Índices orientados a búsquedas y filtrado en listados.

CRUD + roles

- **Admin/Coordinador:** ven todos los clientes.
- **Comercial:** solo clientes asignados (`usuario_id`).
- En alta, `usuario_id` se asigna automáticamente al comercial (si es rol comercial).
- Admin/coordinador pueden asignar o reasignar el comercial desde formularios.
- El servidor fuerza el `usuario_id` correcto para impedir manipulaciones por comerciales.

Seguridad y validación

- Tokens **CSRF** en alta/edición/borrado.
- Sanitización básica y validaciones de formato.
- Control de **DNI duplicado** antes de insertar/actualizar.

Borrado protegido

- Si existen inmuebles ligados al cliente, el borrado se bloquea y se devuelve un error controlado.

3.6.5. Gestión de inmuebles (backoffice + catálogo público)

Objetivo funcional

- Backoffice: alta, edición, listado, filtrado y baja lógica.
- Área pública: catálogo de inmuebles "publicables" según criterios de visibilidad.
- Regla de negocio: cada inmueble pertenece a un cliente/propietario y debe respetar la cartera del comercial.

3.6.5.1. Implementación base (MVC)

- Modelo Inmueble alineado con el esquema real (`ref` , `propietario_id` , `comercial_id` , `direccion` , `localidad` , `provincia` , `cp` , `tipo` , `operacion` , `precio` , `estado` , `activo` , `archivado` , `fecha_alta` , etc.).
- Controladores para área admin y pública.
- Vistas: listado + formulario en admin, y estructura del catálogo público.
Rutas registradas en el Router.

3.6.5.2. Criterios de publicación (catálogo público)

Un inmueble se considera publicable cuando cumple simultáneamente:

- -
 -
- ```
estado =
'activo'
activo =
1
archivad
o = 0
```

### **3.6.5.3. Resolución de bloqueo técnico (404) — Actualización 07/12/2025**

- **Router:** normalización de separadores en Windows ( \ → / ) antes del procesamiento para que la ruta coincidiera con las registradas.
- **Sesiones:** unificación de claves de sesión ( user\_id vs id\_usuario ) en controladores, alineado con AuthController .
- **Roles:** acceso habilitado también para **comercial**, aplicando restricciones específicas.

### **3.6.5.4. Mejora de navegación contextual (Return Path)**

- Implementación de return\_to con validación ( validateReturnTo() ) para aceptar solo rutas internas /admin/\* y evitar **open redirect**.
- Persistencia en un campo hidden y redirección tras guardar manteniendo el contexto (ficha de cliente).

### **3.6.5.5. Imagen principal del inmueble (subida segura)**

- Migración: 04\_add\_imagen\_to\_inmuebles.sql (columna opcional imagen ).
- Directorio dedicado: /public/uploads/inmuebles + .htaccess para desactivar ejecución PHP y deshabilitar listado.
- Validaciones: move\_uploaded\_file() , tamaño ≤ 2 MB, MIME real ( finfo\_file() ), getimagesize() , dimensiones ≤ 1920×1920, nombre único y limpieza de imagen anterior.

### **3.6.5.6. Refuerzo de seguridad por rol (cartera comercial)**

- Filtrado por rol en listados y validación en controladores para impedir que un comercial gestione inmuebles fuera de su cartera.

### **3.6.5.7. Carrusel de propiedades destacadas (Home)**

- Reutiliza criterio de “publicable”.
- Selección aleatoria estable 24h con ORDER BY RAND(TO\_DAYS(CURDATE())) .
- Frontend sin librerías externas (CSS scroll-snap + JS nativo).

## **3.6.6. Landing pública “Vende” (captación de propietarios)**

- Ruta pública: GET /vende
- Controlador: HomeController::vende()
- Vista: app/views/vende/index.php
- Estilos específicos: public/assets/css/landing.css
- Reutiliza endpoints existentes ( /contacto y /tasacion ) para mantener *single source of truth*.

## **3.6.7. Cumplimiento normativo (RGPD y cookies)**

- Páginas legales provisionales bajo /legal/\* con controlador dedicado y vistas en app/Views/legal/ .  
Banner de cookies con aceptar/rechazar y preferencia guardada en localStorage ( cookie\_consent ).

### **3.6.8. Gestión de interfaz y UX**

- Vistas parciales en `app/Views/partials/` para componentes reutilizables.
- Visibilidad controlada desde `HomeController` mediante banderas (`$showHero` , `$mostrar_tarjeta` , etc.), respetando MVC.

### **3.6.9. Módulo de demandas (CRM)**

- Tabla demandas relacionada con clientes y usuarios (comerciales).
- Campo `características` en JSON (serialización/deserialización a array).
- Control por rol: admin/coordinador ven todo; comercial solo su cartera.
- Integración en ficha de cliente y listado global `/admin/demandas` .
- Validación: CSRF, coherencia de rangos y normalización de JSON a [] .

## **3.7. Manejo de errores (resumen)**

---

Se implementó un manejador global de excepciones (`set_exception_handler`) en el punto de entrada. En producción:

- Los detalles técnicos se registran en el log del servidor.
- Se muestra un mensaje genérico y amigable al usuario final.
- Se evita fuga de información sensible ante fallos de BD o errores no controlados.

## **3.8. Justificación de decisiones técnicas**

- **¿Por qué PDO?** facilita una futura migración de motor y soporta sentencias preparadas (clave contra inyección SQL).
- **¿Por qué password\_hash ?** estándar de la industria (PASSWORD\_DEFAULT) con salt incorporado y coste computacional alto contra fuerza bruta.
- **¿Por qué uniqid en nombres de archivo?** evita colisiones y reduce riesgos de sobrescritura/ataques por nombres conocidos.
- **¿Por qué encapsular subidas con control de errores?** se evita error fatal y se mantiene una UX controlada.

## **3.9. Estado actual del proyecto**

---

La plataforma es funcional y estable en sus módulos principales:

- Autenticación por roles, gestión de usuarios, clientes y auditoría.
- Tasación online con Google Sheets como fuente de datos y envío de correos corporativos.
- Gestión de inmuebles (backoffice + catálogo público) con reglas de cartera comercial, return path e imagen principal.
- Demandas (CRM) integradas con clientes y control por rol.
- Front público de propiedades y buscador rápido (Hero Search).
- Landing "Vende" orientada a captación de propietarios.
- Despliegue productivo en Sered y respaldo en Debian 13 (Cloudflare Tunnel + Tailscale para administración privada).

## 4. Evaluación y conclusiones finales

---

### 4.1. Indicadores de garantía de calidad

---

Para asegurar el éxito del proyecto, se han definido y validado estos indicadores:

- **Seguridad:** pruebas manuales de inyección SQL y XSS en formularios públicos, con apoyo de IA para revisar casos.
- **Usabilidad:** validación por parte de un usuario externo (Tutor comercial) del flujo de alta de inmuebles.

### 4.2. Procedimientos de evaluación y incidencias

---

Se registró como incidencia la diferencia de comportamiento entre Windows y Linux en rutas (mayúsculas/minúsculas y separadores), lo que provocaba fallos de carga y 404. Se resolvió normalizando rutas y ajustando el autoloader para respetar el casing correcto, dejando documentada la lección aprendida para despliegues futuros.

### 4.3. Conclusiones

---

El desarrollo de este proyecto ha permitido consolidar conocimientos de arquitectura MVC en PHP, seguridad web (CSRF, validación en servidor, control por roles) y despliegue real en entornos diferentes (producción y respaldo). El resultado es una aplicación funcional y coherente con la operativa de una agencia inmobiliaria: gestión interna (CRM) y parte pública de catálogo.

Como indicador de éxito, el proyecto ha despertado interés para su implantación en un entorno real: una agencia que actualmente trabaja con WordPress ha valorado positivamente la solución y ha planteado su adopción. La implantación se realizará de forma progresiva, con apoyo técnico durante el proceso (tareas de programador y soporte informático) hasta completar la puesta en marcha.

### 4.4. Líneas futuras y mejoras previstas

---

Aunque el proyecto se entrega plenamente funcional y con los requisitos académicos cubiertos, se han identificado mejoras evolutivas propias de un entorno real, donde los requisitos de UI/UX y la operativa cambian según el cliente y el uso diario.

#### **Evolución de interfaz (UI/UX):**

- Ajustes estéticos y de usabilidad en base a feedback real (jerarquía visual, textos, CTAs, accesibilidad y consistencia de componentes).
- Refinado del diseño responsive (espaciados, tipografía y comportamiento del menú en móvil) para adaptarse a necesidades del cliente y a métricas de uso.

#### **Herramienta de tasación (migración de datos a BBDD):**

- Sustituir progresivamente consultas basadas en hoja de cálculo por una capa de persistencia en base de datos (MySQL/MariaDB), manteniendo una arquitectura desacoplada
- (Service/Repository).  
Objetivo: mejorar trazabilidad, rendimiento, mantenimiento y permitir gestión interna (CRUD de
- tablas maestras:  
zonas, CP, precios/m<sup>2</sup>, ajustes, etc.).  
Añadir cacheado, validación avanzada y registros de auditoría para un uso continuado en la agencia.

Estas mejoras se plantean como continuidad natural del proyecto en un contexto profesional, manteniendo el diseño modular para facilitar su incorporación sin reescrituras completas.

## **5. Referencias**

---

- **PHP Documentation:**<https://www.php.net/docs.php>
- **PSR Standards (PHP-FIG):**<https://www.php-fig.org/psr/>
- **Bootstrap 5 Docs:**<https://getbootstrap.com/docs/5.0/getting-started/introduction/>

## **Anexos**

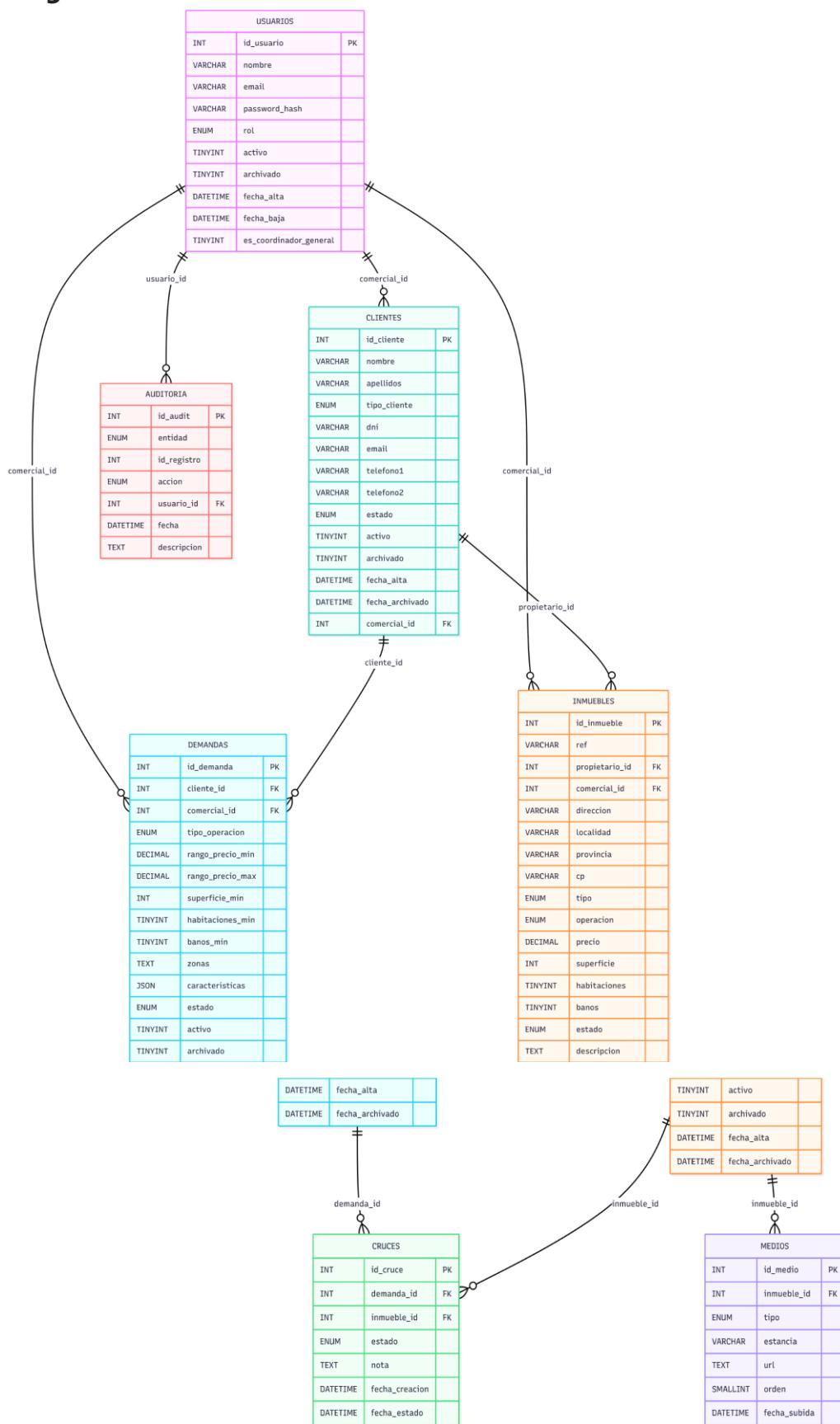
---

### **Anexo A Prototipo Figma Wireframes**

- **Enlace (solo lectura)**[Enlace a Figma](#)
- **Contenido:**Landing (móvil/tablet/escritorio), Listado, Ficha de inmueble, Contacto y Panel Admin.
  - **Archivo Anexo\_A\_Wireframes.pdf**

## **Anexo B — Diagramas técnicos**

## Diagrama de la base de datos



**Figura.** Diagrama del modelo relacional (tablas principales, relaciones e integridad referencial).



# Anexo C — Detalle de módulos implementados

---

## Autenticación y seguridad

**Objetivo:** garantizar un acceso seguro al backoffice y proteger rutas privadas según el rol.

### Funcionalidades y medidas aplicadas

- **Login seguro**
  - Verificación de contraseñas mediante hash ( `password_verify` ) y almacenamiento con `password_hash` .
  - Validaciones en servidor y mensajes de error controlados (sin filtrar si el usuario existe o no).
- **Gestión de sesión**
  - Regeneración del ID tras autenticación ( `session_regenerate_id(true)` ) para evitar *session fixation*.
  - Cookies de sesión con flags **HttpOnly** y **Secure** (este último solo aplica bajo HTTPS).
- **Control de acceso por rol (RBAC)**
  - Restricción de acceso mediante comprobaciones de autenticación y rol (p. ej. `requireAuth()` / `requireRole()` ), aplicadas en controladores para limitar acciones según permisos.
- **Hardening en servidor (Apache / .htaccess)**
  - Reglas de reescritura hacia el Front Controller y bloqueo estricto del acceso público a ficheros sensibles o de control de versiones (p. ej. `.env` , `.git` , `.htaccess` ), devolviendo **403 Forbidden** ante intentos de lectura.  
◦ `public/` se mantiene como único punto expuesto (DocumentRoot), dejando `app/` fuera de acceso directo.

### Archivos implicados (rutas)

- `app/Controllers/AuthController.php`
- `app/Models/User.php`
- `app/views/auth/login.php`
- `app/views/admin/dashboard.php`
- `app/views/layouts/header.php`
- `public/index.php` `public/.htaccess`
- `config/.htaccess` (*si lo usas para blindar config*)
- 

## Herramienta de tasación

**Objetivo:** captación de leads mediante valoración online + automatización de comunicaciones.

### Funcionalidades

- **Formulario interactivo** para valoración de inmuebles, con validación completa en servidor.
- **Envío de correos transaccionales**
  - Email al **cliente** (confirmación / siguiente paso).
  - Email a la **agencia** (lead con los datos).
  - Implementación mediante servicio SMTP centralizado y plantillas HTML reutilizables.
- **Seguridad de entrada**
  - Sanitización estricta de los datos ( `trim` , control de caracteres, etc.).
  - Validación de tipos/formatos (email válido, rangos numéricos coherentes, etc.).
  - Configuración SMTP desde `.env` (sin credenciales hardcoded).

## Archivos implicados (rutas)

- app/Controllers/TasacionController.php
- app/Services/MailService.php
- app/Lib/PHPMailer/\* (*si lo integraste manualmente sin Composer*)
- app/views/tasacion/formulario.php app/views/emails/layout.php
- app/views/emails/tasacion\_cliente.php
- app/views/emails/tasacion\_agencia.php
- public/assets/css/tasacion.css config/config.php
- .env (*no versionado*)
- 

## Gestión de usuarios (CRUD)

**Objetivo:** administrar cuentas internas con control de ciclo de vida, seguridad y trazabilidad.

### Funcionalidades

- **Listado**
  - Visualización de usuarios con filtros por estado/rol.
- **Creación y edición**
  - Formularios validados en servidor (email único, formato correcto, contraseñas robustas, etc.).
- **Baja lógica (Soft Delete)**
  - Desactivación de usuarios sin perder historial ni romper relaciones.
- **Protección anti-suicidio**
  - Un administrador no puede desactivarse/bloquearse a sí mismo para evitar quedarse sin acceso.
- **Fotos de perfil (subida segura)**
  - Validación de tipo **MIME real** (no confiar en `$_FILES['type']` ).
  - Control de tamaño ( $\leq 2$  MB) y manejo de errores `UPLOAD_ERR_*`.
  - Renombrado aleatorio/único y limpieza automatizada de archivos antiguos al reemplazar.
  - Mensajes amigables en UI (sin exponer rutas internas).
- **Sistema de auditoría (flat-file)**
  - Logs de seguridad en fichero de texto para accesos, fallos, bloqueos y eventos críticos, con visor integrado en el panel de administración.
- **Mejora de UX en dashboard**
  - Header y dashboard muestran la foto y el email del usuario logueado para mejorar orientación y feedback visual.

## Archivos implicados (rutas)

- app/Controllers/UserController.php
- app/Models/User.php
- app/views/admin/users/index.php
- app/views/admin/users/create.php
- app/views/admin/users/edit.php
- app/views/layouts/header.php
- app/views/admin/dashboard.php
- database/migrations/01\_add\_foto\_perfil.sql