*PROYECTO*:

# Portal Empleo CyL

*Realizado por:*

*S.G. Dev*

- *Gabriela Scripcariu*

**Desarrollo de Aplicaciones Web (DAW)**

**Valladolid, a 3 de febrero de 2026**

# Contenido

# 1. Introducción

**PortalEmpleoCyL** es una aplicación web diseñada para centralizar y facilitar el acceso a las ofertas de trabajo publicadas por la Junta de Castilla y León. La temática del proyecto se enmarca en la **digitalización de servicios públicos** y el fomento de la empleabilidad regional.

Para su desarrollo, se ha seleccionado el conjunto de datos abierto (**Dataset**) de "Ofertas de Empleo" proporcionado por el Portal de Datos Abiertos de la Junta de Castilla y León. La elección de este dataset responde a la necesidad de ofrecer una interfaz más intuitiva, rápida y filtrable que las consultas tradicionales en boletines oficiales o portales administrativos densos.

**Finalidad del proyecto:** La aplicación aborda la necesidad ciudadana de encontrar oportunidades laborales de forma ágil. A través de la integración de estos datos, los usuarios pueden visualizar las ofertas vigentes, filtrarlas por criterios específicos y acceder a los detalles de cada convocatoria de manera directa, mejorando la transparencia y la eficiencia en la búsqueda de empleo público y privado gestionado por la comunidad autónoma.

*Figura 1. Logotipo app.*

# 2. Análisis

### 2.1. Dataset utilizado

Para el desarrollo de esta aplicación, se ha seleccionado el conjunto de datos abierto proporcionado por la Junta de Castilla y León.

- **Nombre:** Ofertas de Empleo.
- **URL:** Ofertas de Empleo | Datos Abiertos | Junta de Castilla y León
- **Campos utilizados:**
    1. **Identificador (id):** Código único de la oferta que permite gestionar la unicidad de los datos y evitar duplicados en la interfaz.

2. **Título de la oferta:** Nombre descriptivo del puesto de trabajo. Es el campo principal que se muestra en las tarjetas de la aplicación.
3. **Provincia:** Campo categórico (ej. Burgos, Valladolid, León) utilizado para alimentar el motor de filtrado de la aplicación.
4. **Descripción / Contenido:** Bloque de texto que incluye los requisitos, funciones y condiciones del puesto. Se visualiza detalladamente al seleccionar una oferta.
5. **Fecha de publicación:** Dato temporal que permite ordenar las ofertas de forma cronológica descendente (de más reciente a más antigua).
6. **Enlace (url):** Dirección web que redirige al usuario a la convocatoria oficial para formalizar la inscripción.

### 2.2. Requisitos funcionales:

En este apartado se enumeran las funcionalidades que el sistema debe permitir realizar al usuario, organizados por modulo:

1. **RF-01: Registro de usuarios**

- El sistema debe permitir a cualquier visitante registrarse proporcionando nombre, email, contraseña, provincia, sector y nivel de experiencia.
- El email debe ser único en el sistema; si ya existe, se debe mostrar un mensaje de error.
- La contraseña debe tener al menos 6 caracteres.
- La contraseña se debe almacenar cifrada con hash bcrypt (`password_hash`).
- Tras el registro, el usuario debe quedar autenticado automáticamente.

2. **RF-02: Inicio de sesión y cierre de sesión**

- El sistema debe permitir a un usuario registrado iniciar sesión con email y contraseña.
- Las credenciales se validan contra la base de datos usando `password_verify`.
- La sesión se gestiona mediante `$_SESSION` de PHP.
- El usuario debe poder cerrar sesión en cualquier momento, destruyendo la sesión activa.

3. **RF-03: Listado de ofertas con paginación**

- El sistema debe mostrar las ofertas de empleo en un listado paginado (12 ofertas por página).
- Cada oferta debe mostrar: titulo, empresa, provincia y un extracto de la descripción.
- La paginación debe permitir navegar entre páginas de resultados.

4. **RF-04: Búsqueda y filtrado de ofertas**

- El sistema debe permitir buscar ofertas por texto libre (busca en título, descripción y empresa).
- El sistema debe permitir filtrar ofertas por provincia mediante un desplegable.

- Los filtros deben aplicarse de forma asíncrona (AJAX) sin recargar la página completa.
- Los filtros de texto y provincia deben poder combinarse.

**5. RF-05: Detalle de oferta**

- El sistema debe mostrar una vista de detalle con toda la información de una oferta: titulo, empresa, provincia, descripción completa, fecha de publicación y enlace a la fuente original.

**6. RF-06: Sistema de favoritos**

- Un usuario autenticado debe poder guardar ofertas como favoritas.
- Un usuario debe poder eliminar ofertas de sus favoritos.
- Cada favorito debe tener un estado: interesado, aplicado o descartado.
- El usuario debe poder cambiar el estado de un favorito en cualquier momento.
- El sistema debe mostrar un listado con todos los favoritos del usuario.
- Un usuario no puede guardar la misma oferta dos veces.

**7. RF-07: Perfil de usuario**

- El usuario debe poder ver y editar su perfil: nombre, provincia, sector y nivel de experiencia.
- Los datos del perfil se utilizan para personalizar las recomendaciones de la IA.

**8. RF-08: Búsqueda por lenguaje natural con IA**

- El sistema debe permitir al usuario escribir consultas en lenguaje natural (por ejemplo: "Busco trabajo de programador en Valladolid").
- El sistema debe enviar la consulta a la API de Google Gemini, que extrae filtros de búsqueda (texto, provincia) a partir de la consulta.
- Los filtros extraídos se aplican sobre la base de datos de ofertas y se muestran los resultados al usuario.
- Si no hay resultados exactos, el sistema debe hacer búsquedas alternativas (solo por provincia, solo por texto, o mostrar ofertas recientes).

**9. RF-09: Recomendaciones personalizadas con IA**

- El sistema debe permitir al usuario solicitar recomendaciones personalizadas.
- La IA analiza el perfil del usuario (provincia, sector, experiencia) y las ofertas disponibles, y devuelve las 5 ofertas más relevantes con una justificación para cada una.

**10. RF-10: Control de límite de peticiones a la IA**

- Cuando se alcanza el límite de peticiones de la API (error 429), el sistema debe mostrar un mensaje amigable al usuario indicando que espere 1 minuto.
- Durante ese minuto, el sistema debe bloquear el envío de nuevas consultas en el lado del cliente.

**11. RF-11: Mapa interactivo de ofertas**

- El sistema debe mostrar un mapa de Castilla y León con marcadores en cada provincia.

- Cada marcador debe indicar el número de ofertas disponibles en esa provincia.
- Al pulsar un marcador, se debe mostrar un popup con el nombre de la provincia y el total de ofertas.

**12. RF-12: Estadísticas y gráficos**

- La página de inicio debe mostrar estadísticas generales: total de ofertas, numero de provincias y número de empresas.
- Se debe mostrar un gráfico de barras con el número de ofertas por provincia (Chart.js).

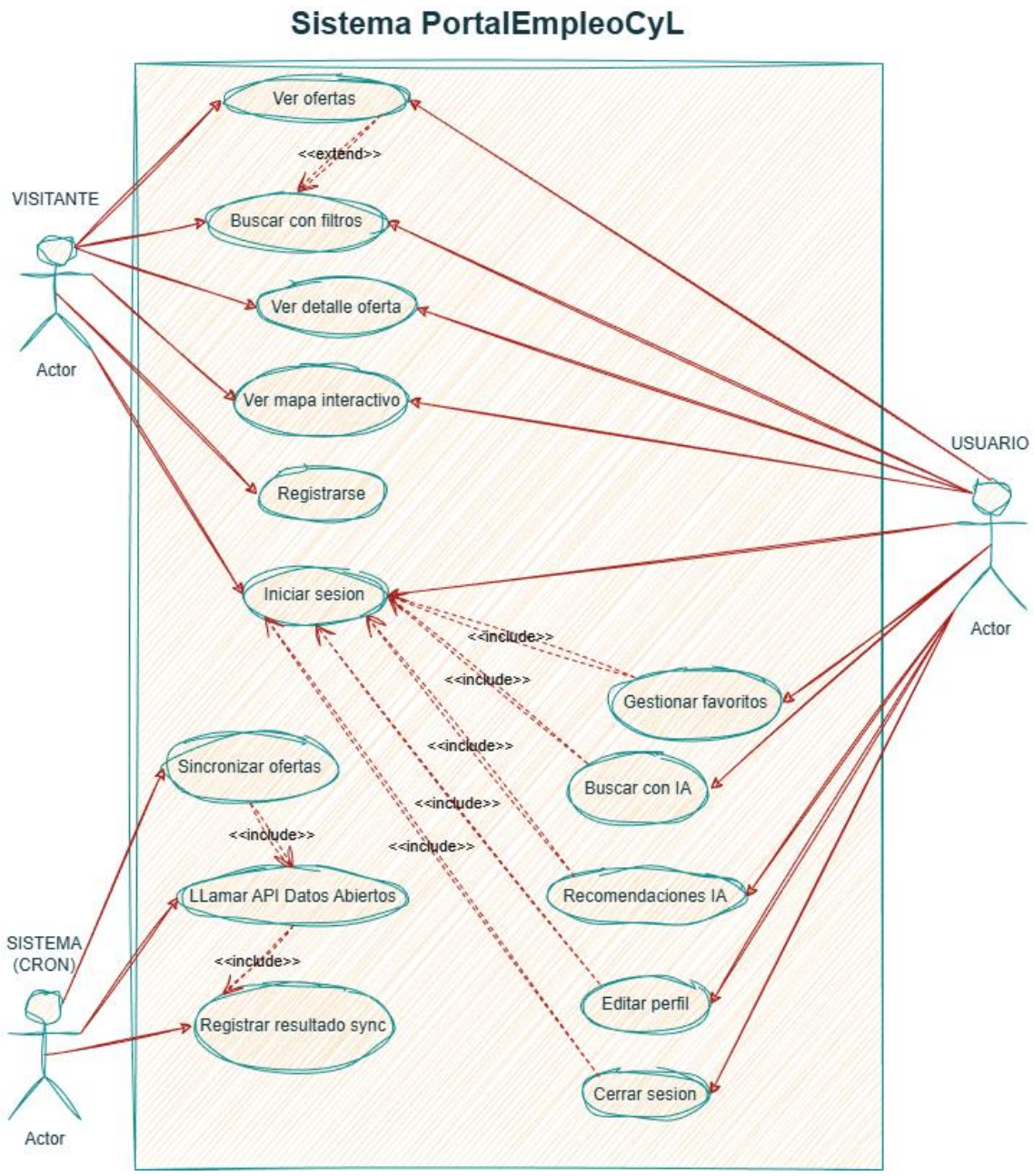
**13. RF-13: Sincronización automática de ofertas**

- El sistema debe sincronizar las ofertas de empleo desde la API de Datos Abiertos de la JCyL cada 24 horas mediante un cron externo (cron-job.org).
- El endpoint de sincronización debe estar protegido por un token de seguridad.
- El sistema debe comprobar si ya se sincronizo en las últimas 24 horas para evitar ejecuciones duplicadas.
- Cada sincronización debe registrar el resultado (ofertas añadidas, actualizadas, estado, errores) en la tabla registros_sincronizacion.

**14. RF-14: Validación de formularios**

- Los formularios de registro y login deben validarse tanto en el cliente (JavaScript) como en el servidor (PHP).
- La validación en cliente debe ser en tiempo real, mostrando mensajes de error junto a cada campo.

### 2.3. Diagramas de casos de uso




El sistema se basa en la interacción del usuario final con la plataforma y en la automatización de la gestión de datos mediante tareas programadas. Los casos de uso identificados son:

**1.Actor: Visitante** (Usuario no registrado que accede a la web. Puede ver ofertas, buscar con filtros, ver el mapa y registrarse.)

**Ver listado de ofertas:**

- El actor accede a la sección "Ofertas".
- El sistema muestra un listado paginado con 12 ofertas por página, ordenadas por fecha de publicación descendente.

- Cada tarjeta muestra título, empresa, provincia y un extracto de la descripción.
- El actor puede navegar entre paginas usando los botones de paginación.

**Buscar ofertas con filtros:**

- El actor escribe un texto en la barra de búsqueda y/o selecciona una provincia en el desplegable.
- El sistema envía una petición AJAX al servidor con los filtros.
- El servidor busca ofertas cuyo título, descripción o empresa coincidan con el texto y/o cuya provincia coincida con la seleccionada.
- El sistema actualiza el listado de ofertas sin recargar la página.

**Ver detalle de oferta:**

- El actor pulsa "Ver oferta" en una tarjeta del listado.
- El sistema muestra una página con todos los datos de la oferta: titulo, empresa, provincia, descripción completa, fecha de publicación y enlace a la fuente original.

**Ver mapa interactivo:**

- El actor accede al mapa desde la página de inicio o la sección de ofertas.
- El sistema carga un mapa de Castilla y León con Leaflet.js.
- Se muestran marcadores en cada provincia con el número total de ofertas.
- El actor pulsa sobre un marcador y se muestra un popup con el nombre de la provincia y el total.

**Registrarse:**

- El actor pulsa "Registro" en la navegación.
- Se muestra el formulario con los campos: nombre, email, contraseña, confirmar contraseña, provincia, sector y nivel de experiencia.
- La validación en tiempo real (JavaScript) comprueba el formato de los campos mientras se escriben.
- El actor pulsa "Registrarse".
- El servidor valida los datos, cifra la contraseña y crea el usuario.
- Se inicia sesión automáticamente y se redirige a la página de ofertas.

**Iniciar sesión:**

- El actor pulsa "Entrar" en la navegación.
- Se muestra el formulario de login con email y contraseña.
- El actor introduce sus credenciales y pulsa "Iniciar sesión".
- El servidor verifica las credenciales con password_verify.

- Se crea la sesión PHP y se redirige a la página de ofertas.

**2.Actor: Usuario registrado** (Usuario autenticado. Además de las acciones del visitante, puede gestionar favoritos, usar el chat con IA, pedir recomendaciones y editar su perfil.)

**Ver listado de ofertas:**

- El actor accede a la sección "Ofertas".
- El sistema muestra un listado paginado con 12 ofertas por página, ordenadas por fecha de publicación descendente.
- Cada tarjeta muestra título, empresa, provincia y un extracto de la descripción.
- El actor puede navegar entre paginas usando los botones de paginación.

**Buscar ofertas con filtros:**

- El actor escribe un texto en la barra de búsqueda y/o selecciona una provincia en el desplegable.
- El sistema envía una petición AJAX al servidor con los filtros.
- El servidor busca ofertas cuyo título, descripción o empresa coincidan con el texto y/o cuya provincia coincida con la seleccionada.
- El sistema actualiza el listado de ofertas sin recargar la página.

**Ver detalle de oferta:**

- El actor pulsa "Ver oferta" en una tarjeta del listado.
- El sistema muestra una página con todos los datos de la oferta: titulo, empresa, provincia, descripción completa, fecha de publicación y enlace a la fuente original.

**Ver mapa interactivo:**

- El actor accede al mapa desde la página de inicio o la sección de ofertas.
- El sistema carga un mapa de Castilla y León con Leaflet.js.
- Se muestran marcadores en cada provincia con el número total de ofertas.
- El actor pulsa sobre un marcador y se muestra un popup con el nombre de la provincia y el total.

**Gestionar favoritos:**

- El actor pulsa el icono de corazón en una oferta para guardarla como favorita (estado: "interesado").
- El sistema envía una petición AJAX POST y registra el favorito.
- El actor accede a "Favoritos" para ver todas sus ofertas guardadas.
- El actor puede cambiar el estado de un favorito a "aplicado" o "descartado".

- El actor puede eliminar un favorito pulsando el icono de eliminar.

**Buscar con IA por lenguaje natural:**

- El actor accede a "Chat IA" en la navegación.
- Escribe una consulta en lenguaje natural, por ejemplo: "Busco trabajo de administrativo en León".
- El sistema envía la consulta a la API de Google Gemini junto con el perfil del usuario y las provincias disponibles.
- Gemini extrae los filtros (texto: "administrativo", provincia: "León") y devuelve un JSON.
- El sistema busca ofertas en la base de datos con esos filtros.
- Se muestran las ofertas encontradas en el chat junto con un mensaje explicativo de la IA.

**Pedir recomendaciones personalizadas con IA:**

- El actor pulsa el botón de recomendaciones en el chat de IA.
- El sistema envía el perfil del usuario y las 20 ofertas más recientes a Google Gemini.
- Gemini analiza la compatibilidad y devuelve las 5 mejores ofertas con una justificación para cada una.
- Se muestran las recomendaciones en el chat con el motivo de cada sugerencia.

**Editar perfil:**

- El actor pulsa su nombre en la navegación y selecciona "Mi Perfil".
- Se muestra el formulario con los datos actuales: nombre, provincia, sector y nivel de experiencia.
- El actor modifica los campos que desee y pulsa "Guardar cambios".
- El servidor valida y actualiza los datos.
- Se muestra un mensaje de confirmación.

**Cerrar sesión:**

- El actor pulsa su nombre en la navegación y selecciona "Cerrar Sesión".
- El sistema destruye la sesión PHP.
- Se redirige a la página de login.

**3.Actor: Sistema (Cron)** (Proceso automático que se ejecuta cada 24 horas mediante cron-job.org para sincronizar las ofertas desde la API de Datos Abiertos.)

**Sincronizar ofertas automáticamente:**

- Cron-job.org realiza una petición GET a cron.php?token=. . . cada 24 horas.
- El sistema verifica el token de seguridad.

- Comprueba si ya se sincronizo en las últimas 24 horas mediante cache.
- Si no se ha sincronizado, ejecuta el script sincronizar_ofertas.php.
- El script descarga las ofertas desde la API de Datos Abiertos de la JCyL en lotes de 100 registros.
- Para cada oferta, comprueba si existe (por id_fuente): si no existe la inserta, si existe la actualiza.
- Se registra el resultado en la tabla registros_sincronizacion.
- Se limpia la cache para que los usuarios vean los datos frescos.
- Se devuelve un JSON con el estado de la sincronización.

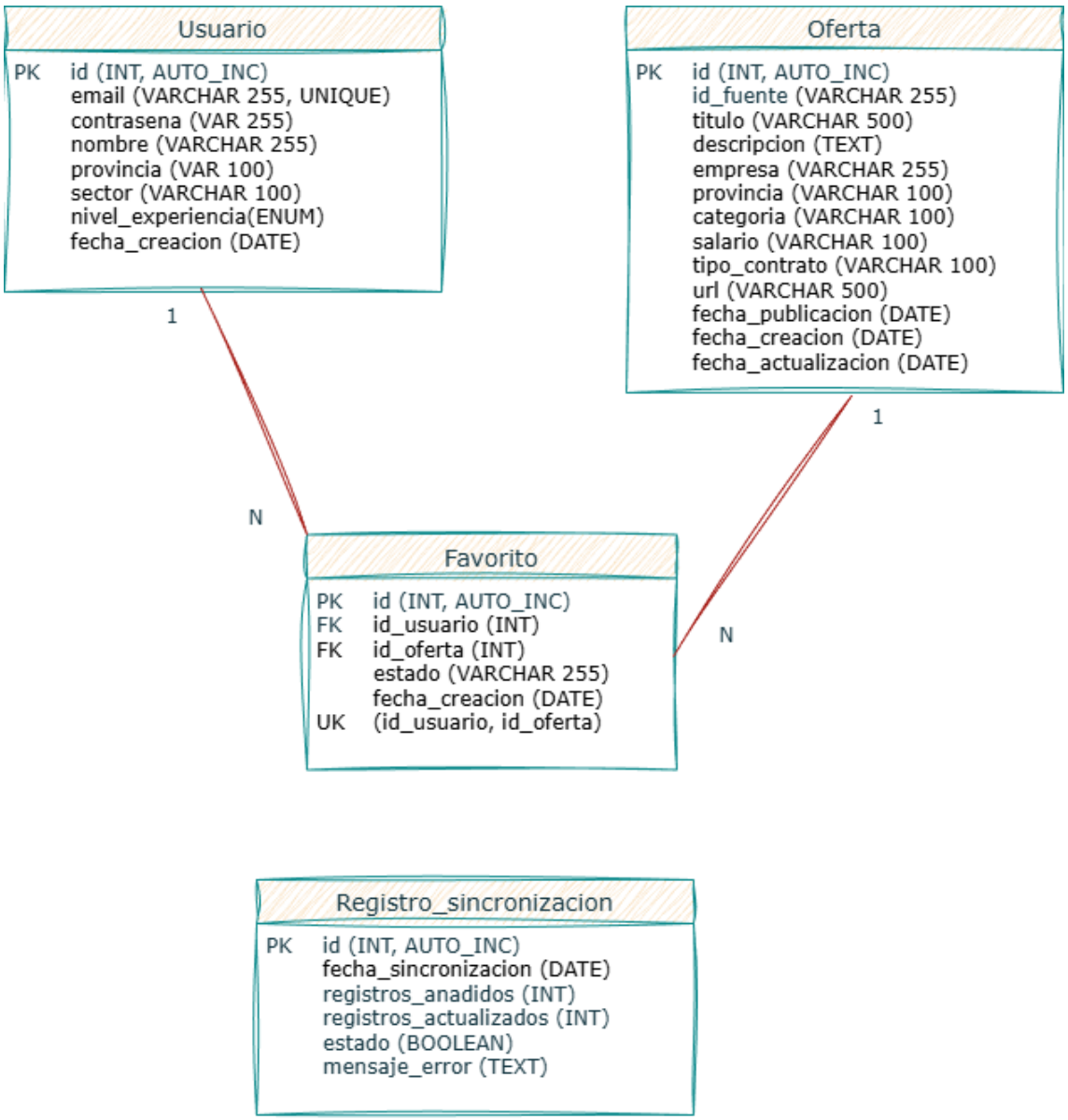
# 3. Diseño

## 3.1. Modelo de base de datos

Se ha diseñado un esquema relacional normalizado para garantizar la integridad de los datos.

La base de datos empleo_cyl utiliza el motor InnoDB con codificación utf8mb4 para soportar caracteres especiales y emojis.

**Diagrama Entidad-Relación:**




**Justificación de tablas y relaciones:**

**Tabla usuario:**

Almacena los datos de los usuarios registrados en la plataforma. Cada usuario tiene un perfil con su provincia, sector profesional y nivel de experiencia, que se utilizan para personalizar las recomendaciones de la IA.

- El campo `email` tiene restricción `UNIQUE` para evitar registros duplicados y un índice para acelerar las búsquedas de login.
- La contraseña se almacena como hash generado con `password_hash()` (algoritmo bcrypt), nunca en texto plano.

- El campo `nivel_experiencia` utiliza un tipo `ENUM` con cuatro valores predefinidos (`sin_experiencia`, `junior`, `intermedio`, `senior`) para garantizar la consistencia de los datos.

**Tabla oferta:**

Tabla central de la aplicación. Almacena las ofertas de empleo descargadas desde la API de Datos Abiertos de la Junta de Castilla y León.

- El campo `id_fuente` almacena el identificador original de la oferta en la API externa, con restricción `UNIQUE` para evitar duplicados durante la sincronización. Esto permite que el proceso de sincronización distinga entre ofertas nuevas (INSERT) y ofertas ya existentes que necesitan actualizarse (UPDATE).
- Se han creado índices en las columnas `provincia`, `categoria` y `fecha_publicacion` porque son los campos más utilizados en los filtros de búsqueda, lo que acelera significativamente las consultas.
- El índice `FULLTEXT` sobre `titulo` y `descripcion` permite realizar búsquedas de texto completo eficientes cuando el usuario busca por palabras clave.
- El campo `fecha actualización` se actualiza automáticamente con `ON UPDATE CURRENT_TIMESTAMP` cada vez que se modifica un registro, permitiendo saber cuándo se actualizo por última vez cada oferta.

**Tabla favorito:**

Implementa la relación muchos-a-muchos entre usuarios y ofertas. Un usuario puede guardar múltiples ofertas y una oferta puede ser guardada por múltiples usuarios.

- Las claves foráneas `id_usuario` e `id_oferta` referencian a las tablas `usuarios` y `ofertas` respectivamente, con `ON DELETE CASCADE` para que al eliminar un usuario o una oferta se eliminen automáticamente sus favoritos asociados, manteniendo la integridad referencial.
- La restricción `UNIQUE KEY favorito_unico (id_usuario, id_oferta)` impide que un usuario guarde la misma oferta dos veces.
- El campo `estado` con tipo `ENUM` permite clasificar cada favorito en tres categorías: `interesado` (guardado para revisar), `aplicado` (ya se ha enviado la candidatura) o `descartado` (no interesa). Esto facilita al usuario organizar su proceso de búsqueda de empleo.

**Tabla registros_sincronizacion (entidad independiente):**

Tabla de auditoría del sistema que registra cada ejecución del proceso de sincronización automática. **Esta tabla no tiene relaciones (foreign keys) con ninguna otra tabla del esquema**, y esto es una decisión de diseño intencionada por las siguientes razones:

- **Función puramente técnica**: No representa una entidad del dominio de negocio (como usuarios, ofertas o favoritos), sino un registro técnico de operaciones del sistema.
- **Independencia temporal**: Los registros de sincronización deben persistir incluso si se eliminan todas las ofertas de la base de datos. Si tuviera una FK hacia `ofertas`, se perdería el historial al limpiar la tabla de ofertas.
- **Patrón de diseño común**: Las tablas de logs, auditoria, métricas y configuración del sistema suelen ser independientes en bases de datos relacionales. No violan ninguna forma normal porque sus atributos son autocontenidos y no dependen de otras entidades.
- **Datos autocontenidos**: Cada registro almacena toda la información necesaria para entender que ocurrió (fecha, contadores, estado, errores) sin necesidad de hacer JOINs con otras tablas.

## 3.2. Diseño de la interfaz

- **Prototipo**:

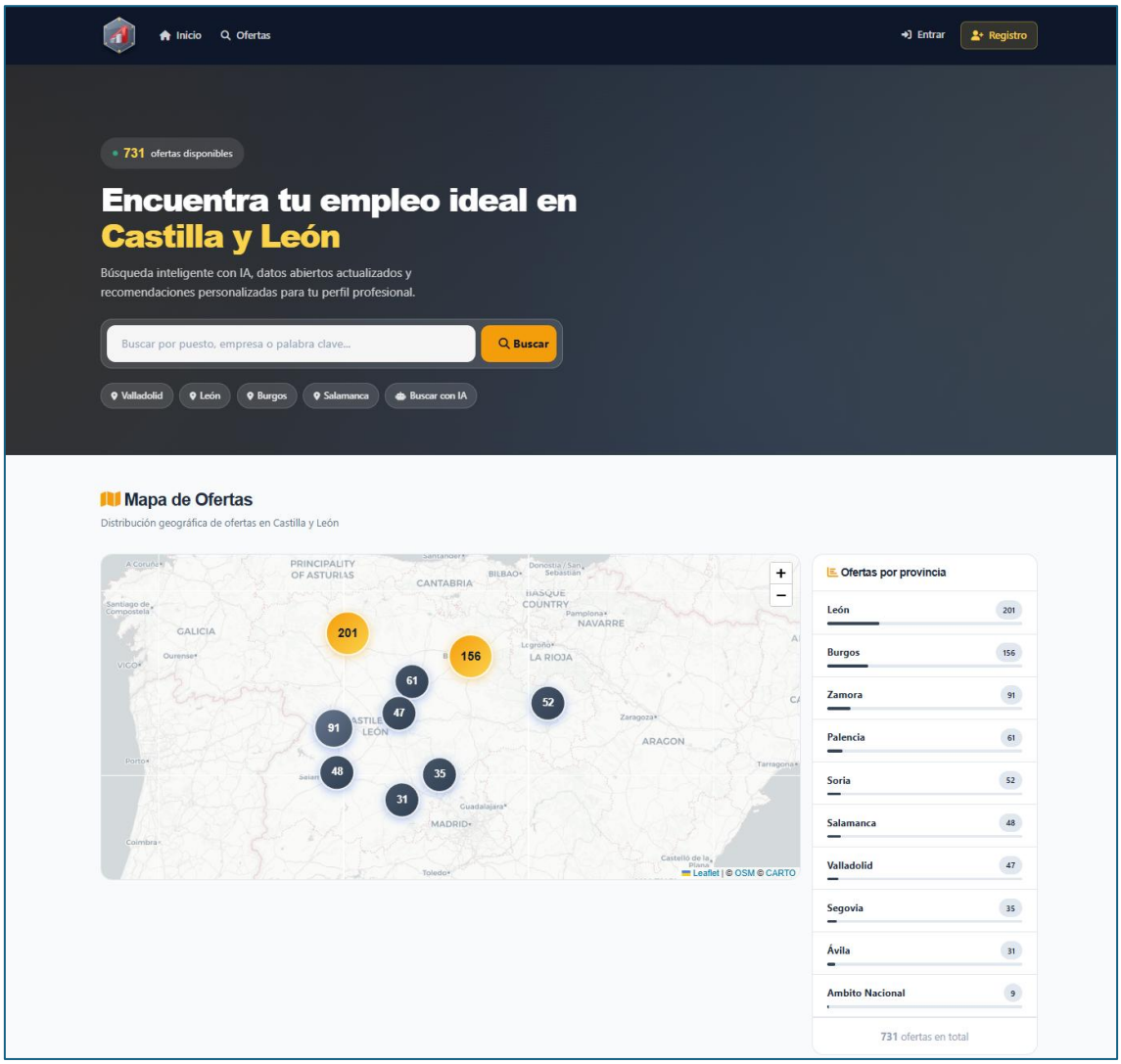



*Figura 1: Desktop: >= 768px (grid multi-columna, navegación completa)*

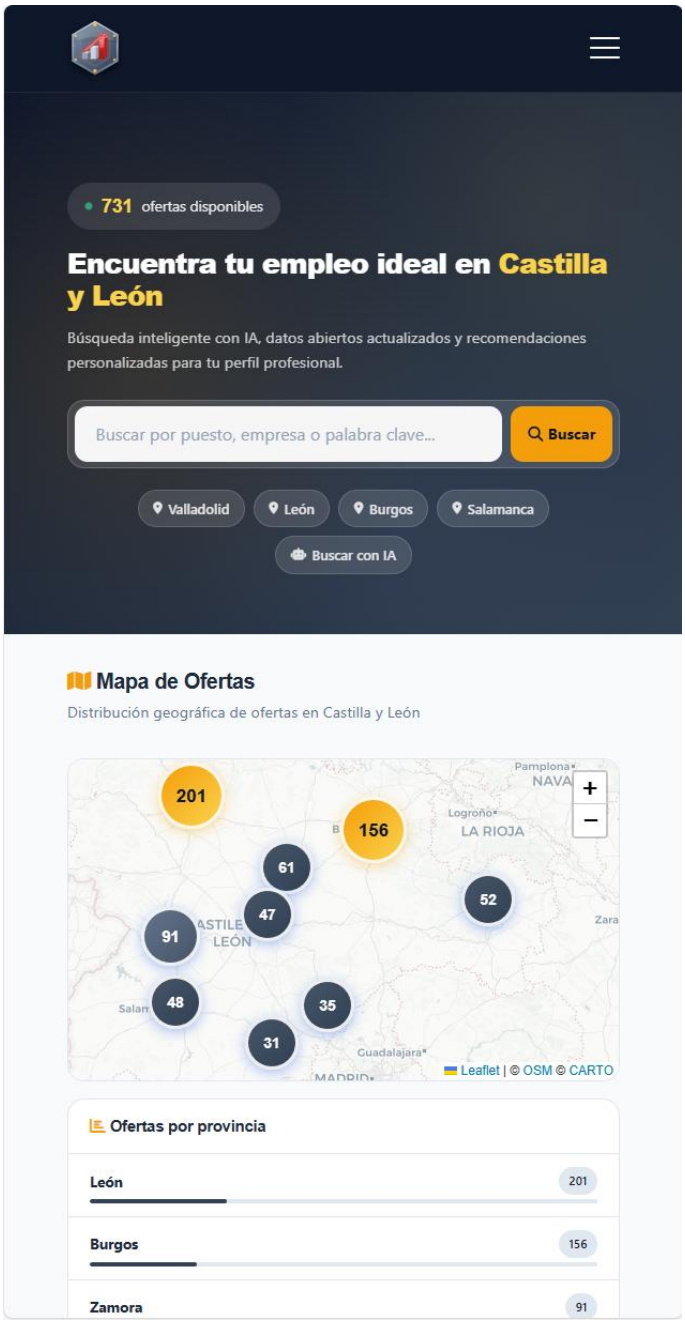



*Figura 2: Mobile: < 768px (1 columna, menú hamburguesa)*

- **Guía de estilos**:

**Paleta de colores:**

| Nombre | Código Hex | Uso |
|---|---|---|
| **Azul profundo** | #0F172A | Fondo navbar, degradados oscuros |
| **Azul institucional** | #1E293B | Texto principal, cabeceras |
| **Azul medio** | #334155 | Elementos secundarios |
| **Azul vibrante** | #475569 | Hover sobre elementos |
| **Azul claro** | #64748B | Texto secundario |
| **Azul suave** | #E2E8F0 | Bordes, separadores |
| **Azul fondo** | #F1F5F9 | Fondo de secciones alternas |
| **Dorado (acento)** | #F59E0B | Botones primarios, destacados, CTA |
| **Dorado claro** | #FCD34D | Hover de botones dorados |
| **Dorado suave** | #FEF3C7 | Fondos de alerta informativa |
| **Acento azul** | #3B82F6 | Enlaces, botones secundarios |
| **Fondo principal** | #F8FAFC | Fondo general de la pagina |
| **Fondo tarjeta** | #FFFFFF | Tarjetas, modales, dropdowns |
| **Éxito** | #059669 | Mensajes de éxito, estados positivos |
| **Peligro** | #DC2626 | Errores, estados negativos |
| **Aviso** | #D97706 | Advertencias, límite de API |

**Tipografías:**

| Fuente | Uso | Pesos utilizados |
|---|---|---|
| **Sora** | Títulos, cabeceras, navbar. | 600 (semibold), 700 (bold) |
| **Inter** | Cuerpo de texto, párrafos, botones. | 400 (regular), 500 (médium), 600 (semibold). |

Ambas fuentes se cargan desde Google Fonts.

**Logotipo:**

El logotipo del Portal de Empleo CyL es un isotipo (solo icono, sin texto) diseñado para transmitir crecimiento profesional, modernidad y éxito. Su estética 3D con efectos de luz y sombra le da un aspecto premium y profesional.

Se ha proporcionado en la carpeta del proyecto diferentes versiones del logotipo, para utilizar en fondos claros, oscuros y cualquier otro fondo.

# 4. Desarrollo

## 4.1. Stack tecnológico

- **Entorno:**

| Capa | Tecnología | Versión |
|---|---|---|
| **Lenguaje backend** | PHP | 8.2 |
| **Base de datos** | MariaDB (MySQL) | 10.4.32 |
| **Servidor web** | Apache | 2.4 (XAMPP) |
| **Lenguaje frontend** | HTML5, CSS3, JavaScript ES6+ | - |
| **Framework CSS** | Bootstrap | 5.3.3 |
| **Iconos** | Font Awesome | 6.5.0 |
| **Gráficos** | Chart.js | 4.x (CDN) |
| **Mapas** | Leaflet.js | 1.9 (CDN) |
| **API de IA** | Google Gemini (gemini-2.0-flash) | v1beta |
| **Datos abiertos** | API Datos Abiertos JCyL | v2.1 |
| **Cron externo** | cron-job.org | - |
| **Hosting** | InfinityFree | - |

- **Software:**

| Programa | Uso |
|---|---|
| **Visual Studio Code** | Editor de código principal |
| **XAMPP** | Servidor local de desarrollo (Apache + MariaDB) |
| **phpMyAdmin** | Gestión de base de datos |
| **Git** | Control de versiones |
| **GitHub** | Repositorio remoto del código |
| **WinSCP** | Cliente FTP para despliegue en hosting |
| **Claude Code (CLI)** | Asistente IA como apoyo al desarrollo |
| **Navegador (Chrome/Edge)** | Pruebas y depuración |

### 4.2. Estructura de carpetas

```
PortalEmpleoCyL/
|
|-- aplicacion/
|   |-- controladores/
|   |   |-- ControladorAutenticacion.php
|   |   |-- ControladorFavoritos.php
|   |   |-- ControladorIA.php
|   |   |-- ControladorInicio.php
|   |   |-- ControladorOfertas.php
|   |   |-- ControladorUsuario.php
|   |
|   |-- modelos/
|   |   |-- Favorito.php
|   |   |-- Oferta.php
|   |   |-- Usuario.php
|   |
|   |-- nucleo/
|   |   |-- BaseDatos.php
|   |   |-- Cache.php
|   |   |-- Configuracion.php
|   |   |-- Controlador.php
|   |   |-- Enrutador.php
|   |
|   |-- vistas/
|       |-- autenticacion/
|       |   |-- login.php
|       |   |-- registro.php
|       |-- ofertas/
|       |   |-- chat-ia.php
|       |   |-- detalle.php
|       |   |-- listado.php
|       |   |-- mapa.php
|       |-- plantillas/
|       |   |-- principal.php
|       |   |-- 404.php
|       |-- usuario/
|       |   |-- dashboard.php
|       |   |-- favoritos.php
|       |   |-- perfil.php
|       |-- inicio.php
|
|-- base_datos/
|   |-- esquema.sql
|
|-- configuracion/
|   |-- base_datos.php
|   |-- claves_api.php
|
|-- publico/
|   |-- css/
|   |   |-- estilos.css
|   |-- img/
|   |   |-- logo.svg
|   |   |-- logo-blanco.svg
|   |   |-- logo-icono.svg
|   |-- js/
|   |   |-- aplicacion.js
|   |   |-- busqueda.js
|   |   |-- chat-ia.js
|   |   |-- validacion.js
|   |-- .htaccess
|   |-- cron.php
|   |-- index.php
|
|-- tareas_programadas/
|   |-- sincronizar_ofertas.php
|
|-- cache/
|-- .gitignore
```

**aplicacion/** — Directorio principal de la lógica de la aplicación. Sigue el patrón de arquitectura MVC (Modelo-Vista-Controlador) para separar la lógica de negocio, el acceso a datos y la presentación en capas independientes.

**aplicacion/controladores/** — Contiene los controladores de la aplicación. Cada controlador es una clase PHP que recibe las peticiones del usuario, interactúa con los modelos para obtener o modificar datos, y devuelve la vista correspondiente.

- **ControladorAutenticacion.php** — Gestiona todo el flujo de autenticación: muestra los formularios de login y registro, valida las credenciales del usuario contra la base de datos usando `password_verify()`, crea la sesión PHP con los datos del usuario y gestiona el cierre de sesión destruyendo la sesión activa.
- **ControladorFavoritos.php** — Maneja las operaciones de ofertas guardadas por el usuario. Permite agregar una oferta a favoritos mediante peticiones AJAX (POST), eliminarla, cambiar su estado entre "interesado", "aplicado" o "descartado", y listar todas las ofertas guardadas del usuario autenticado.
- **ControladorIA.php** — Integra la API de Google Gemini para ofrecer búsqueda inteligente por lenguaje natural. Recibe la consulta del usuario en texto libre, construye un prompt con el contexto del perfil del usuario y las provincias disponibles, envía la petición a Gemini, parsea la respuesta JSON con los filtros extraídos y busca las ofertas coincidentes en la base de datos. También genera recomendaciones personalizadas analizando el perfil del usuario contra las ofertas disponibles.
- **ControladorInicio.php** — Genera la página de inicio de la aplicación. Obtiene las estadísticas generales (total de ofertas, ofertas por provincia), las últimas ofertas publicadas y los datos para los gráficos de Chart.js. Utiliza el sistema de cache para evitar consultas repetidas a la base de datos.
- **ControladorOfertas.php** — Controlador más extenso de la aplicación. Gestiona el listado paginado de ofertas con filtros (texto, provincia), la búsqueda AJAX que actualiza los resultados sin recargar la página, la vista de detalle de cada oferta individual, la vista del mapa interactivo y el endpoint que devuelve los datos geolocalizados de ofertas por provincia para Leaflet.js.
- **ControladorUsuario.php** — Gestiona el perfil del usuario autenticado. Muestra el formulario de edicion con los datos actuales (nombre, provincia, sector, nivel de experiencia) y procesa las actualizaciones validando los datos antes de guardarlos. También genera el dashboard con las estadísticas personales del usuario.

**aplicacion/modelos/** — Contiene las clases que representan las entidades del dominio y encapsulan todo el acceso a la base de datos. Cada modelo utiliza métodos estáticos que llaman a la clase BaseDatos para ejecutar consultas preparadas con PDO.

- **Favorito.php** — Modelo que gestiona la tabla `favoritos`. Implementa métodos para agregar un favorito (INSERT con restricción de unicidad usuario-oferta), eliminar un favorito por ID, cambiar el estado de un favorito y obtener todos los favoritos de un usuario con los datos completos de cada oferta mediante JOIN.
- **Oferta.php** — Modelo principal de la aplicación. Implementa la paginación eficiente con LIMIT/OFFSET, la búsqueda con filtros múltiples usando consultas dinámicas (construye la cláusula WHERE según los filtros activos), la obtención de provincias disponibles con cache de 15 minutos para reducir consultas, y las estadísticas globales agrupadas por provincia para los gráficos.
- **Usuario.php** — Modelo que gestiona la tabla `usuarios`. Implementa la creación de usuarios con hash seguro de contraseña (`password_hash` con PASSWORD_DEFAULT), la verificación de credenciales para el login, la búsqueda por ID y por email, la actualización del perfil y la comprobación de email duplicado antes del registro.

**aplicacion/nucleo/** — Clases fundamentales que forman el framework propio de la aplicación. Son las clases base de las que dependen todos los controladores, modelos y vistas.

- **BaseDatos.php** — Clase que implementa el patrón Singleton para mantener una única conexión PDO a la base de datos durante toda la petición. Ofrece tres métodos principales: `ejecutar()` para INSERT/UPDATE/DELETE con consultas preparadas, `consultar()` para SELECT que devuelve múltiples filas, y `consultarUno()` para SELECT que devuelve una sola fila. La configuración de PDO incluye modo de excepciones, fetch asociativo y desactivación de emulación de prepared statements para mayor seguridad.
- **Cache.php** — Sistema de cache basado en archivos que almacena resultados de consultas frecuentes en formato JSON. Cada entrada tiene un tiempo de vida configurable (por defecto 15 minutos). Cuando se solicita un dato cacheado, comprueba si el archivo existe y no ha expirado; si ha expirado, lo elimina y devuelve false para que se vuelva a consultar la base de datos. Esto reduce las consultas a la base de datos en aproximadamente un 80% en las páginas más visitadas (estrategia Green Coding).
- **Configuracion.php** — Clase estática que centraliza la configuración global del sistema. Define constantes como la ruta base del proyecto, el nombre de la aplicación, el número de ofertas por página y el tiempo de sesión. También proporciona métodos para obtener la configuración de la base de datos y las claves de API desde archivos externos.
- **Controlador.php** — Clase base abstracta de la que heredan todos los controladores. Proporciona métodos comunes: `renderizar()` para cargar una vista dentro del layout principal pasándole variables, `redirigir()` para redirigir a otra ruta, `responderJSON()` para devolver respuestas JSON en

peticiones AJAX, `obtenerPost()` para obtener datos POST sanitizados con `htmlspecialchars`, y `requiereAutenticacion()` para proteger rutas que necesitan un usuario logueado.

- **Enrutador.php** — Sistema de enrutamiento que mapea URLs amigables a controladores y métodos. Registra rutas GET y POST con su controlador y método asociado. Cuando llega una petición, busca la ruta coincidente, instancia el controlador correspondiente, y ejecuta el método. Si no encuentra la ruta, muestra la página 404. También soporta parámetros dinámicos en la URL (por ejemplo, `ofertas/ver/{id}`).

**aplicacion/vistas/** — Contiene todas las plantillas PHP que generan el HTML de la interfaz. Cada vista recibe variables desde su controlador y las muestra al usuario.

- **autenticacion/login.php** — Formulario de inicio de sesión con campos de email y contraseña. Muestra mensajes de error si las credenciales son incorrectas. Incluye enlace al formulario de registro.
- **autenticacion/registro.php** — Formulario de registro con validación en tiempo real (JavaScript). Campos: nombre, email, contraseña, confirmar contraseña, provincia (desplegable), sector y nivel de experiencia. Muestra errores de validación del servidor si los hay.
- **ofertas/chat-ia.php** — Interfaz de chat conversacional con la IA. Tiene un área de mensajes con scroll donde aparecen las conversaciones entre el usuario y el asistente, un campo de texto para escribir consultas, un botón de enviar y un botón de recomendaciones personalizadas. Los mensajes se renderizan dinámicamente con JavaScript.
- **ofertas/detalle.php** — Muestra toda la información de una oferta individual: titulo, empresa, provincia, descripción completa, fecha de publicación y enlace a la fuente original. Incluye botón para añadir a favoritos y botón de volver al listado.
- **ofertas/listado.php** — Listado de ofertas en formato de tarjetas (cards) con paginación. Incluye barra de búsqueda y filtro por provincia que actualizan los resultados por AJAX sin recargar la página. Cada tarjeta muestra título, empresa, provincia, un extracto de la descripción y botón para ver el detalle.
- **ofertas/mapa.php** — Vista que contiene un mapa interactivo de Castilla y León usando Leaflet.js. Muestra marcadores en cada provincia con el número de ofertas disponibles. Al pulsar un marcador se muestra un popup con el nombre de la provincia y el total de ofertas.
- **plantillas/principal.php** — Layout maestro de la aplicación. Define la estructura HTML común: cabecera con etiquetas meta y enlaces a CSS, barra de navegación responsive con logo, enlaces de navegación y menú de usuario, área de contenido donde se inyecta cada vista, footer, y carga de scripts JavaScript. Todos los demás ficheros de vistas se renderizan dentro de este layout.

- **plantillas/404.php** — Pagina de error que se muestra cuando el usuario accede a una ruta que no existe. Muestra un mensaje amigable y un enlace para volver a la página de inicio.
- **usuario/dashboard.php** — Panel personal del usuario que muestra estadísticas de su actividad: número de ofertas guardadas, ofertas por estado (interesado, aplicado, descartado) y las últimas ofertas añadidas a favoritos.
- **usuario/favoritos.php** — Listado de todas las ofertas que el usuario ha guardado. Cada entrada muestra los datos de la oferta, el estado actual (interesado/aplicado/descartado) con un selector para cambiarlo, y un botón para eliminar el favorito. Las acciones se ejecutan por AJAX.
- **usuario/perfil.php** — Formulario de edicion del perfil del usuario. Permite modificar nombre, provincia, sector y nivel de experiencia. Los datos del perfil influyen en las recomendaciones que genera la IA.
- **inicio.php** — Página principal de la aplicación. Contiene una sección hero con barra de búsqueda rápida, tarjetas con estadísticas generales, un grid con las ofertas más recientes, un gráfico de barras de Chart.js mostrando ofertas por provincia, y un mapa interactivo de Leaflet.js con la distribución geográfica de las ofertas.

**base_datos/** — Directorio que contiene los scripts SQL del proyecto.

- **esquema.sql** — Script SQL completo para crear la base de datos desde cero. Define las cuatro tablas de la aplicación: `usuarios` (datos de registro y perfil), `ofertas` (ofertas de empleo con índices para búsqueda eficiente incluyendo FULLTEXT), `favoritos` (relación muchos-a-muchos entre usuarios y ofertas con estado) y `registros_sincronizacion` (historial de sincronizaciones automáticas con contadores y estado).

**configuracion/** — Archivos de configuración sensibles separados del código fuente.

- **base_datos.php** — Devuelve un array asociativo con los parámetros de conexión a MySQL: host, puerto, nombre de la base de datos, usuario, contraseña y charset. Se utiliza tanto por la clase BaseDatos como por los scripts de sincronización.
- **claves_api.php** — Contiene las claves de las APIs externas utilizadas (Google Gemini). Este archivo está excluido del repositorio Git mediante .gitignore para proteger las credenciales. Devuelve un array con la clave, el modelo y la URL del endpoint.

**publico/** — Directorio raíz accesible desde el navegador (document root). Todos los archivos que el usuario final puede solicitar directamente están aquí.

- **css/estilos.css** — Hoja de estilos principal de la aplicación. Define el sistema de diseño completo mediante variables CSS (custom properties): paleta de colores,

tipografías, sombras, bordes y transiciones. Contiene los estilos de todos los componentes: navbar con glassmorphism, tarjetas de ofertas, formularios de autenticación, chat de IA, mapa, gráficos, paginación, menú hamburguesa responsive y animaciones.

- **img/** — Directorio que contiene los recursos gráficos de la aplicación, principalmente el logotipo en sus diferentes versiones.
- **img/logo.svg** — Version principal del logotipo en formato SVG vectorial. Diseñado para fondos claros. Representa un hexágono con un gráfico de barras 3D ascendente y una flecha de tendencia, simbolizando crecimiento profesional. Colores: gris slate (#64748B a #0F172A) y rojo vibrante (#EF4444 a #991B1B) con acentos dorados.
- **img/logo-blanco.svg** — Version del logotipo optimizada para fondos oscuros (navbar, footer). Misma estructura que el logo principal, pero con degradados más claros para garantizar el contraste sobre superficies oscuras.
- **img/logo-icono.svg** — Version simplificada del logotipo en 32x32 pixeles para uso como favicon, icono de aplicación o espacios reducidos donde el logo completo no sería legible. Mantiene los elementos esenciales: hexágono, barras y flecha.
- **js/aplicacion.js** — Script principal que gestiona la interactividad general: toggle de favoritos con peticiones fetch, inicialización de gráficos de Chart.js, comportamiento del menú hamburguesa, animaciones de scroll y feedback visual de acciones del usuario (toasts/notificaciones).
- **js/busqueda.js** — Modulo dedicado a la búsqueda de ofertas. Implementa la búsqueda en tiempo real por AJAX: cuando el usuario escribe en el campo de búsqueda o cambia el filtro de provincia, envía una petición fetch al servidor y actualiza el listado de ofertas sin recargar la página.
- **js/chat-ia.js** — Lógica del chat de inteligencia artificial. Gestiona el envío de consultas al endpoint de la IA, la recepción y renderizado de respuestas con las ofertas encontradas, las recomendaciones personalizadas, el control de límite de peticiones (muestra aviso cuando se alcanza la cuota de la API y bloquea envíos durante 1 minuto) y la interfaz de mensajes tipo chat.
- **js/validacion.js** — Validación de formularios en el lado del cliente. Comprueba en tiempo real el formato del email, la longitud mínima de la contraseña, la coincidencia entre contraseña y confirmación, y que los campos obligatorios estén completos. Muestra mensajes de error junto a cada campo y desactiva el botón de envió si hay errores.
- **.htaccess** — Archivo de configuración de Apache que gestiona la reescritura de URLs para que todas las peticiones pasen por index.php (patron Front Controller).
- **cron.php** — Endpoint protegido por token que ejecuta la sincronización de ofertas. Es llamado automáticamente cada 24 horas por el servicio externo

cron-job.org. Antes de ejecutar la sincronización, comprueba mediante el sistema de cache si ya se sincronizo en las últimas 24 horas para evitar ejecuciones duplicadas (Green Coding). Devuelve una respuesta JSON con el resultado.

- **index.php** — Punto de entrada único de la aplicación (Front Controller). Inicia la sesión PHP, carga las clases del núcleo, define todas las rutas disponibles (GET y POST) mapeándolas a sus controladores y métodos, y llama al enrutador para resolver la petición actual.

**tareas_programadas/** — Scripts que se ejecutan de forma automática para tareas de mantenimiento del sistema.

- **sincronizar_ofertas.php** — Script principal de sincronización. Contiene la clase `ServicioSincronizacion` que descarga ofertas de empleo desde la API de Datos Abiertos de la Junta de Castilla y León en lotes de 100 registros. Para cada oferta, comprueba si ya existe en la base de datos (por `id_fuente`): si no existe la inserta, si existe la actualiza. Registra los resultados (ofertas añadidas, actualizadas, errores) en la tabla `registros_sincronizacion`. Tras finalizar, limpia la cache si hubo cambios para que los usuarios vean los datos actualizados.

**cache/** — Directorio donde se almacenan los archivos temporales de cache en formato JSON. Cada archivo tiene un nombre basado en el hash MD5 de la clave de cache y una extensión `.cache`. Este directorio está excluido del repositorio Git.

**.gitignore** — Define los archivos y directorios que Git debe ignorar: claves de API, archivos de cache, configuración de IDEs, archivos del sistema operativo y dependencias externas.

### 4.3. Descripción del funcionamiento

El acceso a la plataforma comienza en una página de inicio dinámica que ofrece una visión global del mercado laboral en Castilla y León a través de un mapa interactivo. Para disfrutar de la experiencia completa, el usuario debe dirigirse al apartado de registro, donde tras completar su perfil profesional con datos como sector y experiencia validados al instante por el sistema, se inicia sesión automáticamente mediante un sistema de gestión de identidades seguro.

Una vez autenticado, el ciudadano puede navegar por un listado de ofertas paginado que permite localizar vacantes mediante una barra de búsqueda inteligente y filtros por provincia. Gracias a la tecnología AJAX, estos resultados se actualizan sin esperas ni recargas de página, permitiendo acceder al detalle completo de cada oferta, donde se incluye la descripción técnica y el enlace directo a la fuente oficial de la Junta de Castilla y León.

La interacción avanzada se centraliza en el chat con Inteligencia Artificial, donde el usuario puede conversar en lenguaje natural con un asistente basado en Google Gemini. Este sistema no solo extrae filtros automáticos de las frases del usuario, sino que es capaz de generar recomendaciones personalizadas analizando su perfil profesional. Para garantizar la estabilidad del servicio, el sistema monitoriza las peticiones a la API, gestionando las esperas necesarias si se alcanza el límite de consultas.

Finalmente, el usuario puede realizar un seguimiento activo de su búsqueda guardando vacantes en su sección de favoritos, donde es posible organizar cada oferta según su estado de gestión: interesado, aplicado o descartado. Toda esta información se mantiene actualizada gracias a un proceso de sincronización automatizado que, de forma invisible para el usuario, conecta cada 24 horas con la API de datos abiertos para renovar el catálogo de empleo, apoyándose en un sistema de caché para optimizar el rendimiento del servidor.

# 5. Pruebas

**Usabilidad**: Se centra en la experiencia del usuario, la eficiencia y la claridad de la interfaz.

**Diseño Adaptativo (Responsive):** La interfaz se ajusta perfectamente a móviles, tablets y ordenadores, reorganizando los elementos para mantener la comodidad de lectura y clic.


*Figura 1: Vista móvil.*

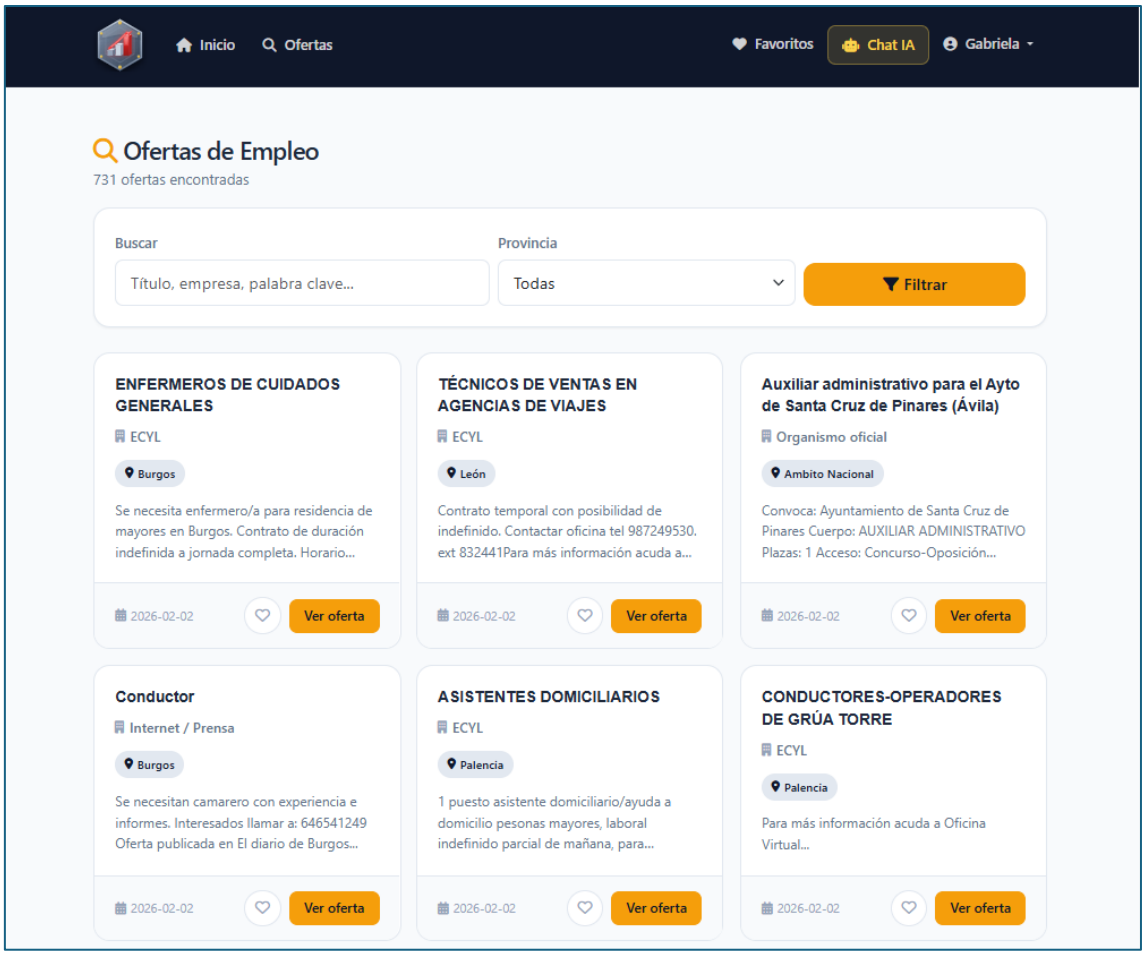



*Figura 2: Vista escritorio.*

**Retroalimentación Inmediata:** El sistema responde al usuario en tiempo real (ej. validación de formularios antes de enviar, cambio de color en botones al pasar el ratón).

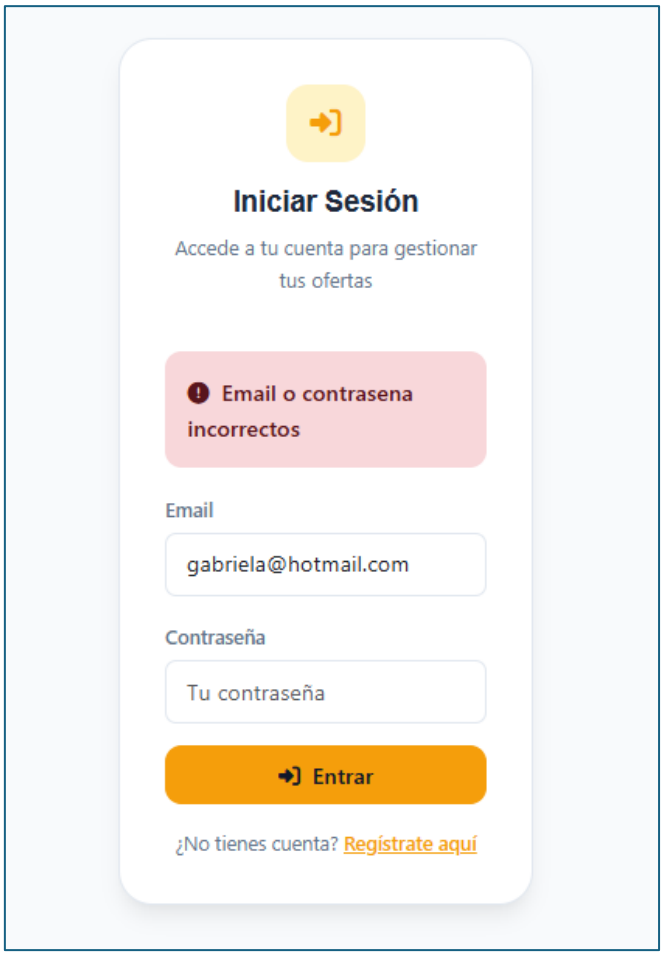



*Figura 3: Validación del formulario al iniciar sesión.*

- **Jerarquía Visual:** Uso de tipografías y colores para destacar lo importante (botones de acción principal en dorado, información secundaria en gris).
- **Prevención de Errores:** Mensajes de ayuda en el registro y avisos claros cuando falta un dato, evitando la frustración del usuario.

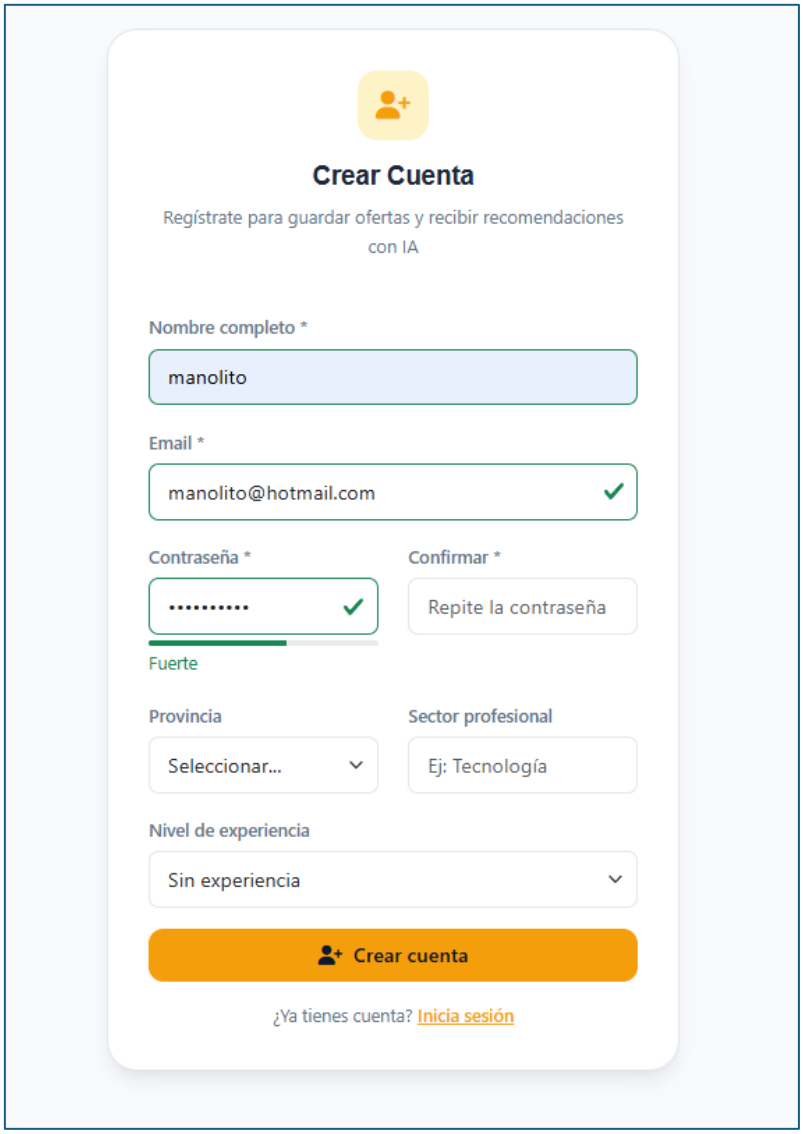



*Figura 4: Mensaje de ayuda en el registro (contraseña correcta, email correcto).*

**Accesibilidad:** Se centra en la eliminación de barreras mediante el cumplimiento de la norma WCAG 2.1.

- **Navegación por Teclado:** Implementación de estilos de foco (`outline`) para que usuarios que no usan ratón sepan dónde están al pulsar la tecla `TAB`.

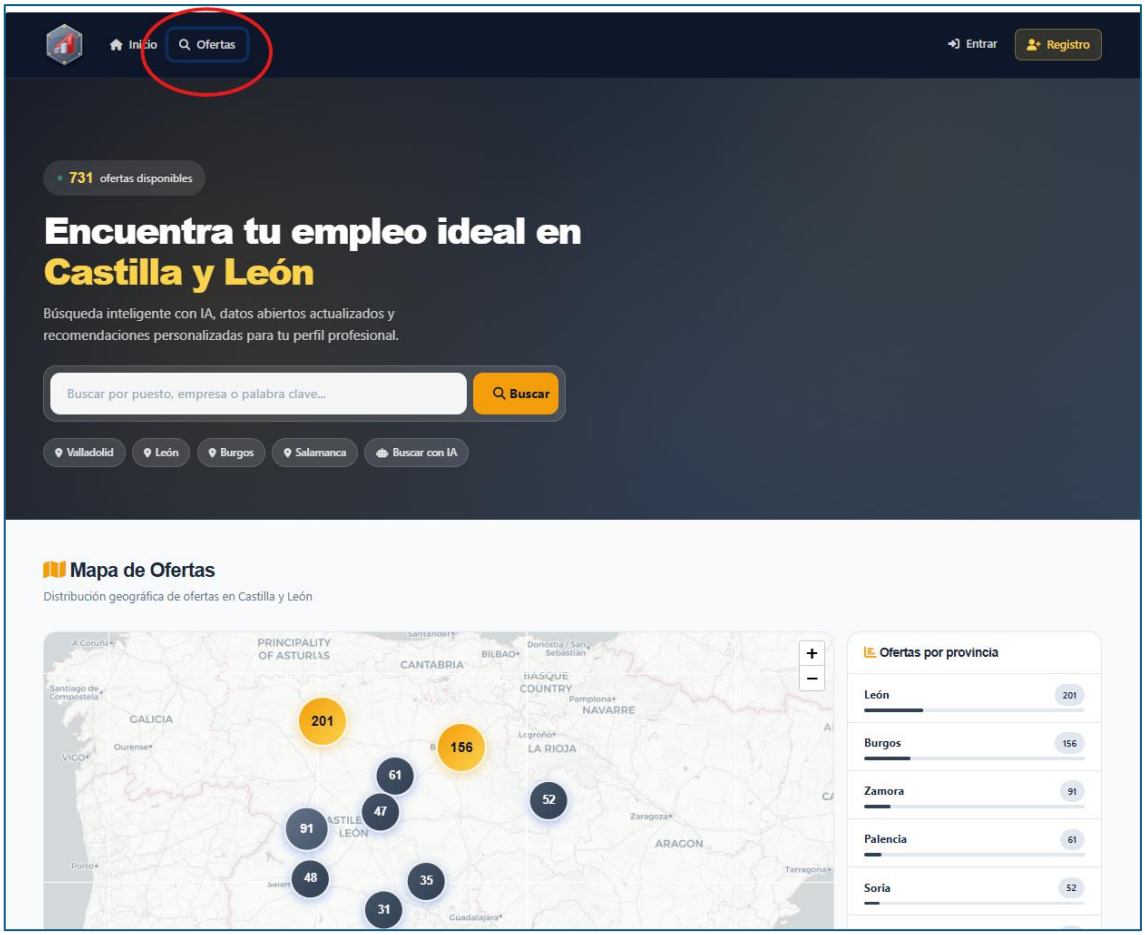



*Figura 5: Recuadro del TAB.*

- **Semántica y ARIA:** Uso de HTML5 puro y etiquetas ARIA para que los lectores de pantalla describan correctamente botones que solo tienen iconos (como el de "Favoritos" o el menú móvil).

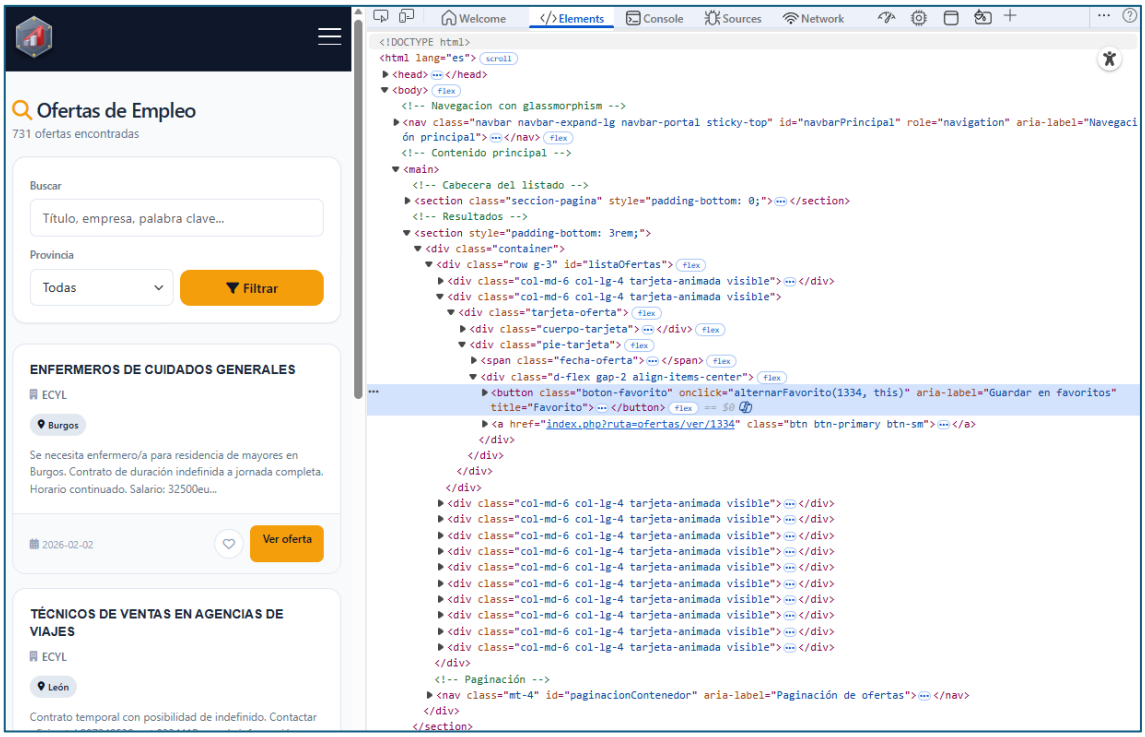



*Figura 6: Inspeccionar botón-favorito.*

- **WCAG Contrast Checker:** para verificar que la relación de contraste entre el texto y el fondo cumple con los niveles de conformidad AA exigidos por la normativa internacional. Como se observa en la captura, la gran mayoría de los elementos principales (títulos de ofertas, textos descriptivos y botones de acción) presentan ratios superiores a 4.5:1, e incluso algunos alcanzan el nivel AAA con ratios superiores a 14:1. Esto garantiza que el contenido sea legible para personas con baja visión o en condiciones de iluminación adversas."

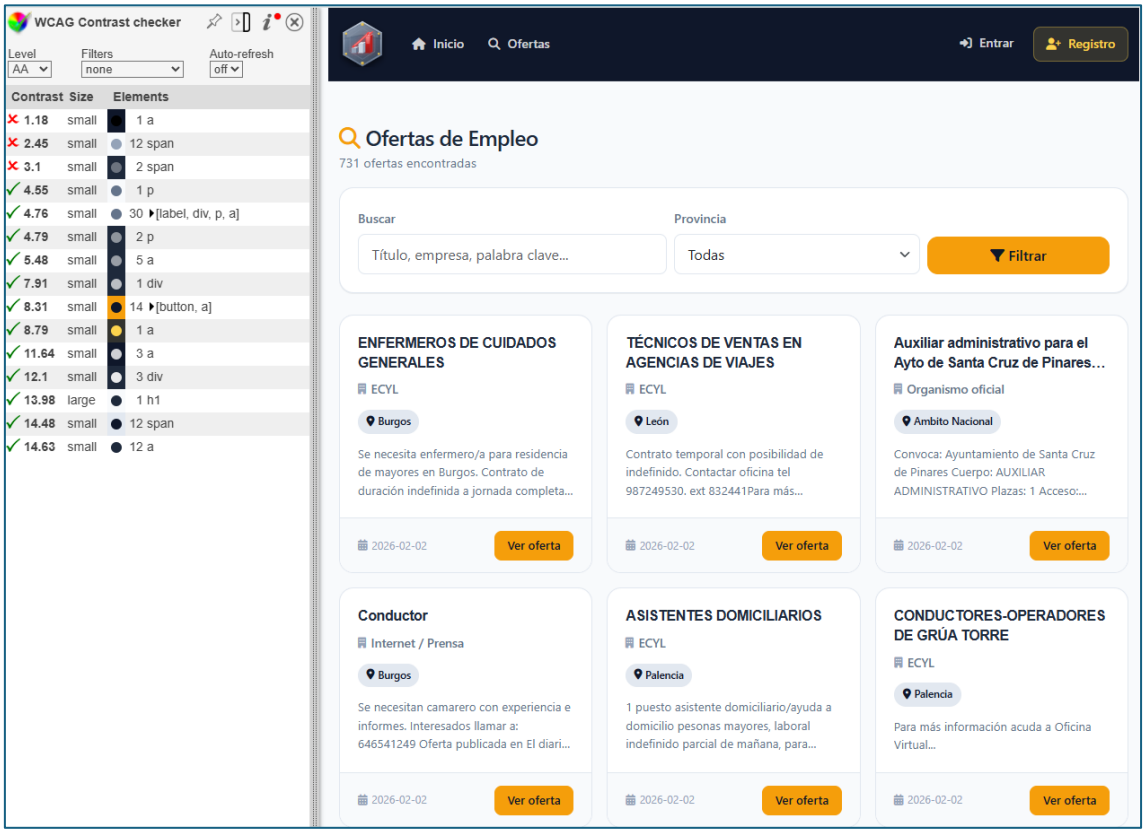



*Figura 7: extensión WCAG.*

**WAVE (WebAIM): Resultado de accesibilidad con WAVE,** la auditoría refleja 0 errores críticos y una excelente puntuación de 8.1/10. Se valida técnicamente el uso de atributos ARIA en los campos de texto, el correcto etiquetado de formularios y una estructura semántica robusta. Esto garantiza que el acceso a la cuenta sea totalmente inclusivo y compatible con lectores de pantalla.

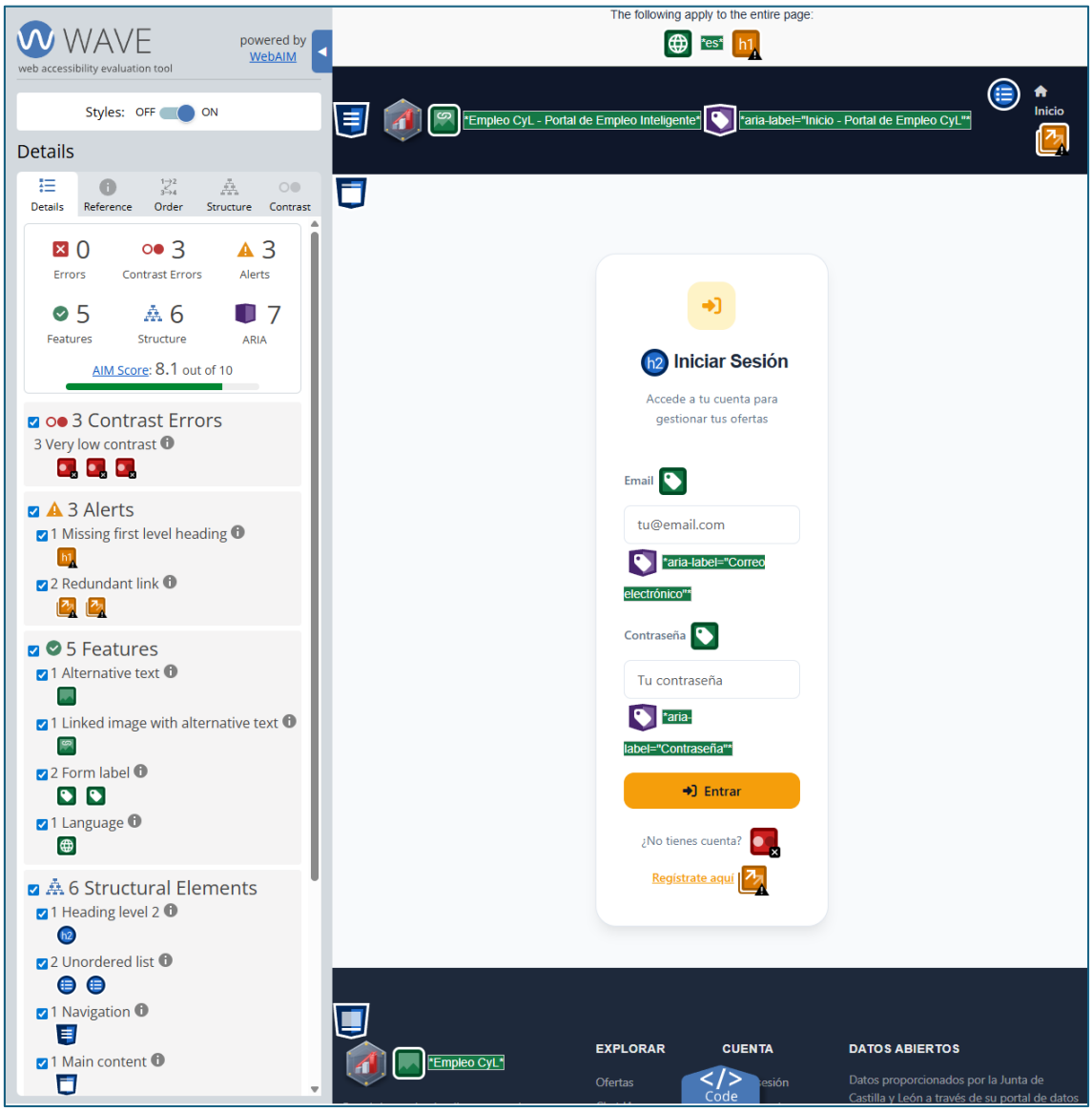



*Figura 8: extensión WAVE.*

# 6. Despliegue

**Instrucciones de instalación en local:**

**Requisitos previos**: XAMPP instalado (Apache + MySQL/MariaDB + PHP 8.x).

**Paso 1** — Clonar el repositorio

```
cd C:\xampp\htdocs
git clone https://github.com/gabrielaSc99/PortalEmpleoCyL.git
```

**Paso 2** — Crear la base de datos

- Abrir XAMPP e iniciar Apache y MySQL.
- Acceder a phpMyAdmin: `http://localhost/phpmyadmin`
- Crear una nueva base de datos llamada `empleo_cyl` con cotejamiento `utf8mb4_unicode_ci`.
- Seleccionar la base de datos creada, ir a la pestaña "Importar" y cargar el archivo `base_datos/esquema.sql`.

**Paso 3** — Configurar la conexión a base de datos

Editar el archivo `configuracion/base_datos.php` con los datos del entorno local:

```
return [
    'host' => 'localhost',
    'puerto' => 3306,
    'nombre_bd' => 'empleo_cyl',
    'usuario' => 'root',
    'contrasena' => '',
    'charset' => 'utf8mb4'
];
```

**Paso 4** — Configurar la clave de la API de IA

Crear el archivo `configuracion/claves_api.php` (no se incluye en el repositorio por seguridad):

```
<?php
return [
    'ia' => [
        'clave' => 'CLAVE_GENERADA_API_KEY',
        'modelo' => 'gemini-2.0-flash',
        'url' =>
'https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-
flash:generateContent'
    ]
];
```

Para obtener la clave, acceder a https://aistudio.google.com/apikeys y crear una API Key.

**Paso 5** — Cargar las ofertas de empleo

Ejecutar la primera sincronización desde el navegador:

```
http://localhost/PortalEmpleoCyL/publico/cron.php?token=aX9kL2mQ7vZ4wP
8nR1jT6yB3cF5hD0sE
```

Este proceso descarga todas las ofertas desde la API de Datos Abiertos de la Junta de Castilla y León y las almacena en la base de datos. Puede tardar varios minutos en la primera ejecución.

**Paso 6** — Acceder a la aplicación

http://localhost/PortalEmpleoCyL/publico/

**Instrucciones de despliegue en Infinityfree:**

**Paso 1** — Crear cuenta en InfinityFree

- Entrar en infinityfree.com y registrarse.
- Crear una nueva cuenta de hosting y elegir un subdominio gratuito (ej: `portalempleocyl.rf.gd`).
- Anotar los datos que proporciona el panel: host MySQL, usuario y contraseña.

**Paso 2** — Crear la base de datos

- En el panel de InfinityFree, entrar en **MySQL Databases**.
- Crear una nueva base de datos y anotar su nombre (ej: `if0_XXXXXXX_empleo_cyl`).
- Entrar en **phpMyAdmin** desde el panel.
- Seleccionar la base de datos creada, ir a la pestaña **Importar** y subir el archivo `base_datos/esquema.sql` del proyecto.

**Paso 3** — Subir proyecto al hosting

- En el panel de InfinityFree, entrar en el **File Manager** (Administrador de Archivos).
- Navegar hasta la carpeta `htdocs/`.
- Subir toda la carpeta del proyecto `PortalEmpleoCyL/` dentro de `htdocs/`, manteniendo la misma estructura de carpetas que en local.
- Comprobar que el directorio `cache/` tiene permisos de escritura (755).

**Paso 4** — Configurar los archivos en el hosting

Una vez subidos los archivos, editarlos directamente desde el **File Manager** de InfinityFree:

`configuracion/base_datos.php` - poner los datos de la base de datos de InfinityFree:

```
'host' => 'sql.infinityfree.com',
'nombre_bd' => 'if0_XXXXXXX_empleo_cyl',
'usuario' => 'if0_XXXXXXX',
'contrasena' => 'TU_CONTRASENA',
```

`configuracion/claves_api.php` - poner la clave de la API de IA (se obtiene gratis en console.groq.com):

```
'clave' => 'TU_CLAVE_DE_GROQ',
```

La web será accesible en: https://tu-dominio/PortalEmpleoCyL/publico/

**Paso 5** — Cargar las ofertas de empleo

Para que aparezcan ofertas en la web, hay que ejecutar la sincronización. Abrir esta URL en el navegador:

```
https://tu-dominio/PortalEmpleoCyL/publico/cron.php?token=aX9kL2mQ7vZ4wP8nR1jT6yB3cF5hD0sE
```

Para que se actualicen automáticamente cada día, crear una tarea programada en cron-job.org con esa misma URL.

**Paso 6** — Comprobar que todo funciona

Entrar en la web, registrar un usuario, hacer login y probar la búsqueda de ofertas y el chat de IA.

- **URLs:**
    - gabrielaSc99/PortalEmpleoCyL
    - https://portalempleocyl.infinityfreeapp.com/publico/

# 7. Sostenibilidad y "Green Coding"

El desarrollo del portal se ha guiado por los principios de Green Coding, un enfoque de programación que busca reducir el consumo energético y la huella de carbono del software mediante decisiones de diseño eficientes.

- **Estrategia de Caché:**

La aplicación implementa un sistema de cache basado en archivos para reducir al mínimo las consultas innecesarias a la base de datos y a las APIs externas.

**Funcionamiento técnico:**

La clase `Cache.php` almacena los resultados de consultas frecuentes como archivos JSON en el directorio `/cache/`. Cada entrada de cache se identifica mediante un hash MD5 de su clave y tiene un tiempo de vida configurable.

**Cache en la sincronización:**

El endpoint `cron.php` utiliza una entrada de cache especial (`ultima_sincronizacion`) con un tiempo de vida de 24 horas. Cuando el servicio cron-job.org llama al endpoint, el sistema comprueba esta marca temporal:

- Si ya se sincronizó en las últimas 24 horas, devuelve un JSON con estado "omitido" y no ejecuta ninguna operación.
- Si han pasado más de 24 horas, ejecuta la sincronización completa y actualiza la marca.

Esto evita que múltiples llamadas accidentales o frecuentes al endpoint generen sincronizaciones redundantes.

La sincronización de ofertas con la API de datos abiertos de la Junta de Castilla y León se realizan mediante un cron job configurado directamente en el panel de control del hosting InfinityFree, programado para ejecutarse una vez al día a las 3:00 AM.

- **Optimización de recursos:**

**Carga de librerías externas vía CDN:**

Todas las librerías externas (Bootstrap, Font Awesome, Chart.js, Leaflet.js) se cargan desde CDNs públicos. Esto tiene dos ventajas:

- Los archivos ya estan minificado y comprimidos (gzip) en el CDN.
- El navegador del usuario probablemente ya los tiene en cache de otras webs que usan las mismas librerías, evitando descargas repetidas.

**Carga diferida de scripts:**

Los scripts JavaScript se cargan al final del documento HTML (antes del cierre de `</body>`) en lugar de en el `<head>`. Esto permite que el contenido HTML y CSS se renderice primero, mejorando el tiempo de carga percibido por el usuario. Además, los scripts específicos de cada vista (como `chat-ia.js` o `busqueda.js`) solo se cargan en las páginas que los necesitan, evitando transferir código innecesario.

**Comunicación asíncrona (AJAX):**

Las operaciones de búsqueda, filtrado, gestión de favoritos y chat con IA se realizan mediante peticiones `fetch()` asíncronas. Esto evita recargar la página completa para cada acción, reduciendo:

- La cantidad de HTML transferido (solo se envía y recibe JSON).
- El procesamiento del servidor (no renderiza toda la plantilla, solo devuelve datos).
- El consumo de ancho de banda del usuario.

- **Reflexión:**

**Estimación del ahorro:**

| Métrica | Sin cache | Con cache (15 min) | Reducción |
|---|---|---|---|
| **Consultas SQL/día (página inicio)** | ~400 | ~96 | 76% |
| **Consultas SQL/día (lista provincias)** | ~100 | ~96 | 4% |
| **Sincronizaciones/día** | Ilimitadas | Máximo 1 | >99% |
| **Peticiones a API externa/día** | Cada visita | 1 vez cada 24h | >99% |

# 8. Conclusiones

## 8.1. Autoevaluación

El desarrollo del Portal de Empleo Inteligente de Castilla y León ha sido un proyecto completo que ha abarcado todas las capas de una aplicación web moderna: desde el diseño de la base de datos relacional hasta la integración con APIs externas de inteligencia artificial, pasando por el desarrollo de una interfaz responsive y el despliegue en un servidor de producción.

- **Dificultades encontradas:**

**Integración con la API de IA:**

La mayor dificultad del proyecto fue integrar la API de inteligencia artificial. Se probaron varios proveedores durante el desarrollo:

- **Google Gemini (gemini-2.0-flash):** fue la primera opción considerada como solución definitiva. Sin embargo, el plan gratuito presentó problemas graves de cuota: el límite de peticiones por minuto era extremadamente restrictivo (error 429 Too Many Requests), llegando incluso a devolver cuota 0 en determinadas claves, lo que inutilizaba completamente el servicio. Además, las claves de API expuestas en repositorios públicos eran revocadas automáticamente por Google.
- **Groq (Llama 3.3 70B Versatile):** se eligió solución final. Ofrece un plan gratuito mucho más generoso (14.400 peticiones diarias), no requiere tarjeta de crédito y proporciona respuestas rápidas y de calidad. Su API sigue el formato OpenAI-compatible, lo que facilita una posible migración futura a otros proveedores.

Es importante destacar que, para la implementación del módulo de asistencia inteligente, se ha optado por una **API de uso gratuito con fines estrictamente educativos**. Esta decisión, aunque estratégica para la viabilidad económica del proyecto, conlleva ciertas limitaciones técnicas que deben ser tenidas en cuenta:

- **Modelo de respuesta:** Al utilizar una versión gratuita, las respuestas del chat pueden no alcanzar la complejidad o precisión de los modelos de pago más avanzados. No obstante, cumple con el objetivo didáctico de demostrar la integración de procesamiento de lenguaje natural en una aplicación web real.
- **Experiencia de Usuario (UX):** La existencia de un límite de cuota (Rate Limit) en las llamadas a la API condiciona la fluidez de la conversación. Para evitar que el usuario quede bloqueado por las restricciones del proveedor, se ha implementado un modo fallback: cuando la API de IA no está disponible, el sistema realiza automáticamente una búsqueda directa

en la base de datos, mostrando un aviso informativo al usuario. De esta forma, el chat siempre devuelve resultados útiles independientemente del estado de la API, garantizando así la estabilidad y continuidad del servicio.

- **Enfoque Educativo:** El valor del proyecto reside en la **arquitectura técnica** (conexión mediante controladores, parseo de JSON y construcción de *prompts*), asumiendo que el rendimiento del chat está supeditado a las herramientas gratuitas disponibles actualmente para el desarrollo académico.

**Problemas con el despliegue:**

El despliegue en InfinityFree presento varios problemas:

- La base de datos no se podía importar directamente porque el archivo SQL exportado contenía una vista (`estadisticas_ofertas`) y el hosting gratuito no permite crear vistas (error de permisos CREATE VIEW).
- El archivo SQL exportado desde phpMyAdmin tenía un BOM (Byte Order Mark) que causaba un error de sintaxis al importar. Se tuvo que regenerar el archivo sin BOM.
- El orden de las tablas en el dump SQL causaba errores de claves foráneas. Se solucionó verificando que las foreign key checks estaban desactivadas durante la importación.

**Problemas con Git y control de versiones:**

La gestión del archivo de claves de API requirió atención especial. Fue necesario añadir `configuracion/claves_api.php` al `.gitignore` para evitar exponer las credenciales en el repositorio público, y proporcionar instrucciones claras en la documentación para que cualquier persona que clone el proyecto pueda crear su propio archivo de configuración.

- **Valoración de la metodología.**

Al tratarse de un proyecto desarrollado de forma individual, la coordinación no se ha centrado en la delegación de tareas, sino en una **gestión integral de roles** y una planificación rigurosa del tiempo. He asumido la responsabilidad de todas las fases del proyecto, desde el análisis de requisitos y el diseño de la arquitectura MVC, hasta la implementación del código y el despliegue de la base de datos.

Trabajar de forma autónoma me ha permitido mantener una visión global y coherente de toda la aplicación, facilitando la integración fluida entre el *backend* en PHP, la lógica de la Inteligencia Artificial y la interfaz de usuario.

En conclusión, la ausencia de un equipo multidisciplinar ha sido compensada con una mayor polivalencia, permitiéndome profundizar en cada capa tecnológica del sistema y garantizar la cohesión total del proyecto.

### 8.2. Líneas futuras

- **Propuestas de futuro.**

Para que la aplicación fuera comercializable o se pudiera poner en producción a gran escala, sería necesario implementar las siguientes mejoras:

**Funcionalidades nuevas:**

- **Sistema de notificaciones**: Enviar emails automáticos a los usuarios cuando se publiquen ofertas que coincidan con su perfil (provincia, sector, nivel de experiencia). Esto requeriría integrar un servicio de envío de correo como SendGrid o PHPMailer y crear una tabla de preferencias de notificación.
- **Alertas personalizadas**: Permitir al usuario definir alertas de búsqueda guardadas (por ejemplo, "Programador en Valladolid") y recibir notificaciones cuando aparezcan nuevas ofertas que coincidan.
- **Historial de búsquedas**: Registrar las búsquedas del usuario para ofrecer sugerencias y mejorar las recomendaciones de la IA con el tiempo.
- **Sistema de valoraciones**: Permitir a los usuarios valorar las ofertas y compartir experiencias sobre los procesos de selección.
- **Comparador de ofertas**: Poder seleccionar varias ofertas y compararlas lado a lado.

**Mejoras técnicas:**

- **Autenticación avanzada**: Implementar recuperación de contraseña por email, verificación de cuenta por correo, autenticación en dos factores (2FA) y login con redes sociales (Google, LinkedIn).
- **API REST publica**: Exponer los datos de ofertas mediante una API REST documentada con Swagger/OpenAPI para que otros desarrolladores puedan consumir los datos.
- **Migración a un framework**: Migrar a Laravel o Symfony para aprovechar su ecosistema de paquetes, ORM (Eloquent/Doctrine), sistema de migraciones de base de datos, colas de trabajo y testing integrado.
- **Cache en memoria**: Sustituir la cache en archivos por Redis o Memcached para mejorar el rendimiento en entornos con alto tráfico.

- **Contenedorización con Docker**: Crear un `docker-compose.yml` que levante Apache, MySQL y PHP en contenedores, facilitando el despliegue en cualquier servidor y garantizando la reproducibilidad del entorno.
- **Tests automatizados**: Implementar tests unitarios con PHPUnit para los modelos y tests de integración para los controladores, asegurando que las futuras modificaciones no rompen funcionalidades existentes.
- **CI/CD**: Configurar un pipeline de integración continua con GitHub Actions que ejecute los tests y despliegue automáticamente en producción tras cada merge a la rama principal.

# 9. Bibliografía

Lenguajes y Documentación Oficial

- **PHP Manual.** *The PHP Group*. https://www.php.net/manual/es/
- **PDO (PHP Data Objects).** *The PHP Group*. https://www.php.net/manual/es/book.pdo.php
- **password_hash.** *PHP Manual*. https://www.php.net/manual/es/function.password-hash.php
- **cURL / JSON.** *PHP Manual*. https://www.php.net/manual/es/book.curl.php | https://www.php.net/manual/es/book.json.php
- **MySQL 8.0 Reference Manual.** *Oracle Corporation*. https://dev.mysql.com/doc/refman/8.0/en/
- **MDN Web Docs (JavaScript & Fetch API).** *Mozilla Developer Network*. https://developer.mozilla.org/es/docs/Web/JavaScript

APIs y Datos Externos

- **Groq API Documentación**. API Keys - GroqCloud
- **Portal de Datos Abiertos de Castilla y León.** *Junta de Castilla y León*. https://datosabiertos.jcyl.es/
- **API de Datos Abiertos (Ofertas de empleo).** Dataset oficial

Librerías, Estilos y Mapas

- **Bootstrap 5 Documentación.** https://getbootstrap.com/docs/5.3/
- **Chart.js Documentación.** https://www.chartjs.org/docs/latest/
- **Leaflet.js Documentación.** https://leafletjs.com/reference.html
- **OpenStreetMap & CARTO Basemaps.** https://www.openstreetmap.org/ | https://carto.com/basemaps/
- **Font Awesome Icons.** https://fontawesome.com/icons
- **Google Fonts (Sora & Inter).** https://fonts.google.com/
- **SweetAlert2.** https://sweetalert2.github.io/

Calidad, Accesibilidad y Seguridad

- **WCAG 2.1.** *Web Content Accessibility Guidelines (W3C)*. https://www.w3.org/WAI/WCAG21/quickref/
- **Google Lighthouse & PageSpeed Insights.** https://developer.chrome.com/docs/lighthouse/ | https://pagespeed.web.dev/
- **WAVE Tool.** *WebAIM*. https://wave.webaim.org/

- **Visual Studio Code.** *Microsoft*. https://code.visualstudio.com/
- **Git Documentación & GitHub.** https://git-scm.com/doc | https://github.com/
- **XAMPP & phpMyAdmin.** https://www.apachefriends.org/ | https://www.phpmyadmin.net/
- **InfinityFree & cron-job.org.** https://www.infinityfree.com | https://cron-job.org