BBr – BuscaBares

Realizado por:

BBr Team

- *Adrián Cantalapiedra Fernández*
- *Adrián Castro Meneses*
- *Francisco Santos Guerra*

Desarrollo de Aplicaciones Web (DAW)

Valladolid, a 5 de febrero de 2026

ÍNDICE

1. Introducción

1.1. Contexto del proyecto académico y tecnológico

El presente documento constituye la memoria técnica, funcional y organizativa del proyecto denominado BBR (BuscaBares), desarrollado como pieza central del módulo de Proyecto Intermodular correspondiente al segundo curso del Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web (DAW). Este trabajo no solo representa la culminación de un itinerario formativo, sino que simula un entorno de producción real donde convergen múltiples disciplinas: ingeniería de software, administración de sistemas, diseño de interfaces y ciberseguridad.

En el panorama actual de la ingeniería web, la demanda de perfiles profesionales ha evolucionado desde el desarrollador especializado en una única capa (frontend o backend) hacia el perfil Full Stack con capacidades de DevOps. Las empresas modernas requieren ingenieros capaces de concebir una aplicación desde la base de datos hasta la interfaz de usuario, pero también de entender y gestionar la infraestructura subyacente que soporta dicha aplicación. El proyecto BBR se ha concebido bajo esta premisa integradora, buscando superar la mera implementación de código funcional para adentrarse en la arquitectura de sistemas complejos, la contenerización y la seguridad perimetral avanzada.

El marco académico en el que se inscribe este proyecto exige el cumplimiento riguroso de cuatro Resultados de Aprendizaje (RA) fundamentales: la identificación de necesidades del sector productivo (RA1), el diseño y desarrollo técnico de la solución (RA2), la planificación documental (RA3) y la defensa de los procedimientos adoptados (RA4). BBR responde a estos criterios mediante una arquitectura desacoplada que utiliza Laravel [1] en el backend y Next.js en el frontend, desplegados sobre una infraestructura de virtualización propia basada en Proxmox y Docker, implementando políticas de seguridad de Zero Trust.

1.2. Problema o necesidad detectada en el sector

La industria turística y de hostelería constituye uno de los motores económicos principales de la Comunidad Autónoma de Castilla y León. Esta región, vasta en extensión y rica en patrimonio enológico y gastronómico, genera una cantidad masiva de datos relacionados con su oferta de servicios. La administración pública, a través de la Junta de Castilla y León [2], ha realizado esfuerzos significativos para digitalizar y liberar esta información a través de su portal de datos abiertos (*Open Data*), ofreciendo catálogos de establecimientos, eventos culturales y recursos turísticos.

Sin embargo, se ha detectado una brecha significativa entre la disponibilidad de estos datos y su consumo efectivo por parte del ciudadano o el turista final. Los datos gubernamentales suelen publicarse en formatos brutos ("raw data") como CSV, XML o JSON, diseñados para el procesamiento automatizado y no para la lectura humana directa. Las plataformas existentes a menudo sufren de obsolescencia, interfaces de usuario poco intuitivas o falta de integración entre diferentes tipos de recursos (por ejemplo, la desconexión entre la ubicación de un bar y los restaurantes cercanos).

Además, desde una perspectiva técnica, muchas de las soluciones de terceros que intentan agregar esta información carecen de arquitecturas robustas, exponiendo a menudo los servidores a vulnerabilidades de seguridad básicas o sufriendo problemas de rendimiento debido a una mala gestión de la concurrencia y el escalado. BBR nace para solucionar esta doble problemática: por un lado, democratizar el acceso a la información turística mediante una interfaz moderna y reactiva; y por otro, establecer un estándar de despliegue seguro y eficiente que garantice la integridad y disponibilidad del servicio.

1.3. Justificación del uso de datos abiertos

La elección de basar el núcleo de datos del proyecto en fuentes de *Open Data* responde a criterios de sostenibilidad, legalidad y valor social.

En primer lugar, la reutilización de información del sector público (RISP) es una directriz fomentada por la Unión Europea para impulsar la economía digital. Al consumir los datasets oficiales del **Registro de Turismo de Castilla y León**, el proyecto BBR garantiza que la información mostrada a los usuarios (nombres de establecimientos, direcciones, categorización) posee el sello de veracidad y actualización de la administración competente. Esto elimina la necesidad de crear y mantener una base de datos propia desde cero, lo cual sería insostenible en términos de recursos humanos para un proyecto de esta envergadura.

En segundo lugar, el uso de formatos abiertos estándar permite la interoperabilidad. El sistema diseñado no se limita a "leer" un archivo estático, sino que implementa procesos de ingesta (ETL - Extract, Transform, Load) capaces de adaptarse a las actualizaciones periódicas que la administración realice en sus repositorios. Esto dota al proyecto de una vida útil prolongada y de una capacidad de actualización dinámica.

1.4. Objetivos del proyecto

El desarrollo de BBR se articula en torno a un objetivo principal y una serie de objetivos específicos que cubren las dimensiones funcionales, técnica y de infraestructura.

1.4.1 Objetivo general

Diseñar, desarrollar, documentar y desplegar en un entorno de producción simulado una aplicación web escalable que consuma, procese y visualice datos abiertos de la Junta de Castilla y León relativos a la hostelería. La solución debe demostrar la competencia técnica del alumno en el desarrollo *Full Stack* (backend y frontend separados) y en la administración de sistemas avanzados (virtualización, contenerización y seguridad de red).

1.4.2 Objetivos específicos

Dimensión funcional y de datos:

- Implementar un sistema automatizado de ingesta de datos que consuma ficheros JSON/CSV del portal de Open Data, normalice la información (codificación, tipos de datos) y la persista en una base de datos relacional robusta (**MariaDB**).
- Desarrollar una interfaz de usuario (**UI**) intuitiva y responsiva que permita la búsqueda, filtrado y visualización detallada de cafeterías, bares y restaurantes, mejorando la experiencia de acceso a la información pública.

Dimensión arquitectónica y de desarrollo:

- Construir una **API RESTful** segura utilizando el framework **Laravel**, aplicando patrones de diseño MVC y protegiendo los puntos finales mediante autenticación basada en tokens o cookies seguras (**Sanctum**).
- Desarrollar una *Single Page Application* (SPA) con **Next.js**, aprovechando el renderizado híbrido y la gestión de estado en el cliente para ofrecer una experiencia de navegación fluida y sin recargas.

Dimensión de infraestructura y seguridad (DevOps):

- Diseñar y configurar un entorno de virtualización basado en **Proxmox VE**, utilizando contenedores ligeros (**LXC**) optimizados mediante técnicas de *hardened* inverso para permitir la ejecución anidada de Docker.
- Implementar una arquitectura de red **Zero Trust**, eliminando la exposición de puertos en el router doméstico y gestionando todo el tráfico entrante a través de túneles encriptados (**Cloudflare Tunnel**).
- Asegurar las comunicaciones mediante cifrado **SSL/TLS Strict**, configuración de cabeceras de seguridad y despliegue de un cortafuegos de aplicaciones web (**WAF**) con bloqueo geográfico.

2. Análisis

El análisis de requisitos es la fase crítica donde se traducen las necesidades abstractas del sector y los objetivos académicos en especificaciones técnicas concretas. Este proceso asegura que el producto final no solo sea funcional, sino también viable y evaluable según la rúbrica del ciclo.

2.1. Análisis del Dataset y fuentes de datos

El proyecto se alimenta del **Registro de Turismo de Castilla y León**, disponible en el portal de datos abiertos de la Junta [3]. Se han identificado los siguientes conjuntos de datos clave para la funcionalidad de BBR:

- **Restaurantes:** Contiene información sobre nombre, categoría (tenedores), especialidad, capacidad, dirección y coordenadas geográficas.
- **Cafeterías y bares:** Datos similares, enfocados en la categoría de tazas/categoría especial.

Desafíos de calidad de datos identificados:
El análisis preliminar de los ficheros CSV y JSON reveló inconsistencias típicas en datos administrativos:

1. **Codificación de Caracteres:** Presencia de codificaciones mixtas (ISO-8859-1 vs UTF-8), requiriendo una capa de normalización en el backend durante la ingesta.
2. **Georreferenciación Incompleta:** Algunos registros carecen de coordenadas precisas o utilizan sistemas de referencia obsoletos (ED50 en lugar de ETRS89/WGS84), lo que implica la necesidad de integración con APIs de geocodificación o limpieza manual.
3. **Redundancia:** Registros duplicados debido a actualizaciones administrativas no consolidadas.

2.2. Requisitos funcionales

Los requisitos funcionales (RF) definen el comportamiento observable del sistema y las interacciones del usuario. Se han categorizado según los roles de usuario previstos.

RF-01: Gestión de identidad y acceso

- El sistema debe permitir el registro de nuevos usuarios mediante correo electrónico y contraseña.
- El sistema debe gestionar el inicio de sesión (*Login*) y el cierre de sesión (*Logout*) de forma segura.

- La sesión del usuario debe persistir entre recargas del navegador mediante el uso de cookies seguras y encriptadas (HttpOnly, Secure), gestionadas por el backend.

RF-02: Ingesta y procesamiento de datos

- El sistema debe disponer de comandos de consola (Artisan Commands) capaces de descargar los ficheros de datos abiertos, procesarlos línea a línea y actualizar la base de datos local sin detener el servicio.
- El proceso de importación debe ser transaccional: si ocurre un error a mitad de proceso, la base de datos no debe quedar en un estado inconsistente.

RF-03: Consulta y visualización

- El usuario debe poder visualizar un listado paginado de establecimientos.
- El sistema debe ofrecer filtros dinámicos por provincia, municipio y tipo de establecimiento (Bar, Restaurante, Cafetería).
- Cada establecimiento debe tener una vista de detalle que muestre toda la información relevante disponible en el dataset original.

RF-04: API RESTful

- El backend debe exponer endpoints estandarizados que devuelvan respuestas en formato JSON.
- Las respuestas de la API deben incluir metadatos de paginación y códigos de estado HTTP adecuados (200, 404, 500) para su correcta interpretación por el frontend.

2.3. Diagramas de casos de uso

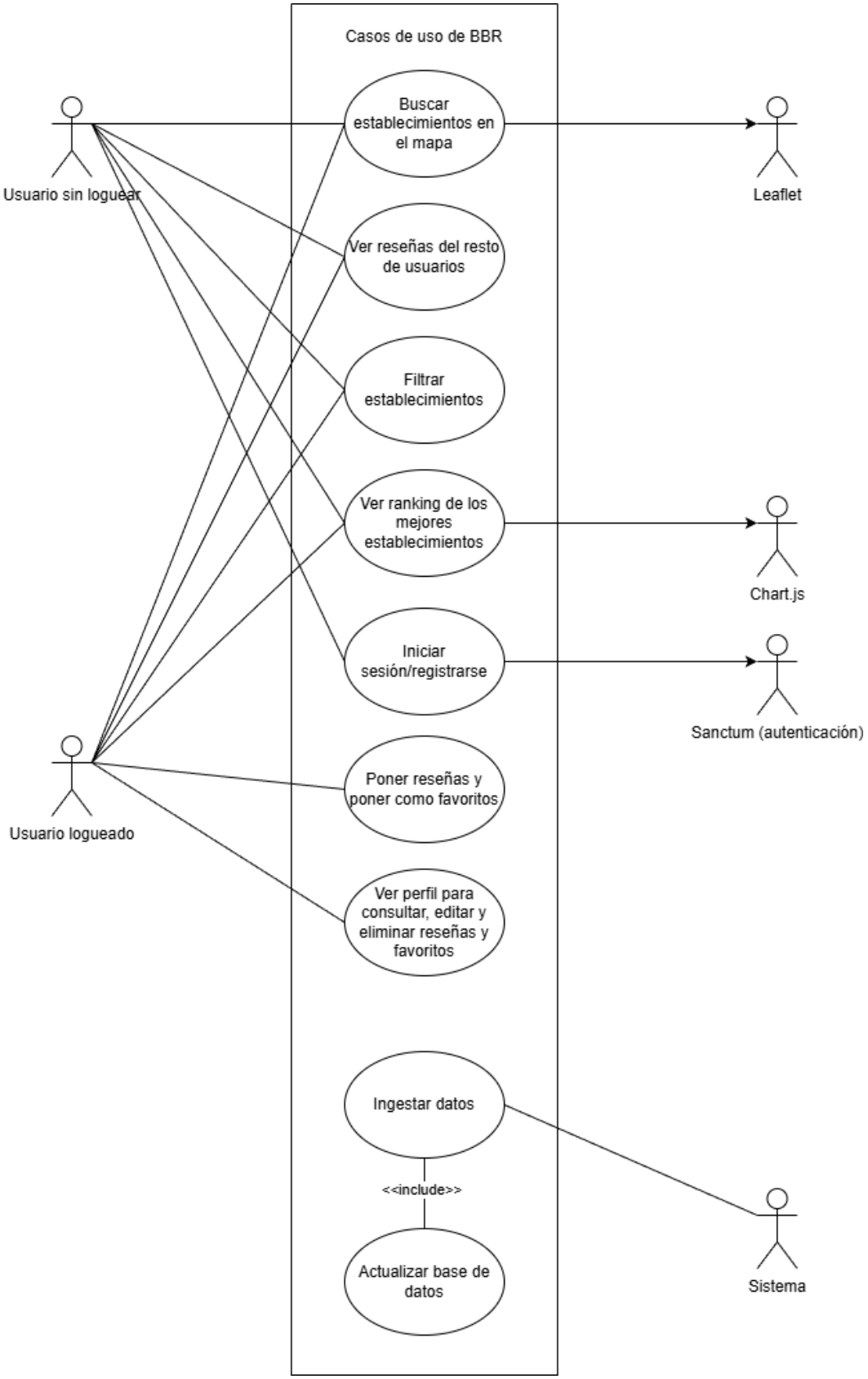

Ilustración 1. Diagrama de casos de uso de BBr

3. Diseño del sistema

El diseño del sistema BBR se aleja de las arquitecturas monolíticas tradicionales para abrazar un enfoque modular y contenerizado, alineado con las prácticas modernas de *Cloud Native* y *Microservicios* (aunque adaptado a la escala del proyecto).

3.1. Arquitectura general

La solución se estructura en tres capas lógicas claramente diferenciadas que se ejecutan en entornos aislados pero interconectados.

1. Capa de presentación (Frontend):

- Implementada como una aplicación independiente en **Next.js**.
- Responsabilidad: Renderizado de la interfaz, gestión del estado del cliente, enrutamiento y comunicación asíncrona con la API.
- Despliegue: Contenedor Docker app-frontend.

2. Capa de aplicación y negocio (Backend):

- Implementada con el framework **Laravel**.
- Responsabilidad: Lógica de negocio, validación de datos, autenticación, autorización y orquestación de la base de datos.
- Despliegue: Contenedor Docker app-backend ejecutando PHP-FPM. Este contenedor es "ciego" al mundo exterior y solo acepta peticiones del servidor web interno.

3. Capa de persistencia (Datos):

- Motor de base de datos **MariaDB**.
- Responsabilidad: Almacenamiento persistente, integridad referencial y transacciones.
- Despliegue: Contenedor Docker database.

Componente transversal de infraestructura: Un servidor **Nginx** interno (app-nginx) actúa como pegamento entre el frontend y el backend. Enruta las peticiones entrantes: si la URL comienza por /api, la dirige al backend (PHP-FPM); si es una petición de navegación web, la dirige al frontend (Node.js/Next.js) o sirve los estáticos compilados. Este diseño resuelve elegantemente los problemas de CORS y unifica el punto de entrada de la aplicación.

3.2. Patrón utilizado

El proyecto utiliza una combinación de patrones arquitectónicos:

- **MVC (Modelo-Vista-Controlador):** En el backend, Laravel implementa estrictamente MVC. Los *Modelos* (Eloquent) gestionan los datos, los

Controladores manejan la lógica de las peticiones API y, en lugar de Vistas tradicionales HTML, devuelven respuestas JSON (serialización de recursos).

- **Arquitectura de componentes:** En el frontend, React/Next.js utiliza una arquitectura basada en componentes reutilizables (Botones, Tarjetas, Barras de Navegación) que encapsulan su propio estado y estilo.

3.3. Diseño de la Base de Datos

El esquema de base de datos relacional se ha diseñado para soportar tanto la lógica de negocio (usuarios) como los datos importados (establecimientos). A continuación, se detallan todas las tablas.

Tabla: bars

#	Nombre	Tipo de datos	Longitud/Co...	Sin signo	Permitir...	Relle...	Predeterminado
1	**id**	BIGINT		☑	☐	☐	AUTO_INCREME...
2	fuente_id	VARCHAR	255	☐	☐	☐	Sin valor predeter...
3	nombre	VARCHAR	255	☐	☐	☐	Sin valor predeter...
4	tipo	VARCHAR	255	☐	☑	☐	NULL
5	provincia	VARCHAR	255	☐	☑	☐	NULL
6	municipio	VARCHAR	255	☐	☑	☐	NULL
7	direccion	VARCHAR	255	☐	☑	☐	NULL
8	lat	DECIMAL	10,8	☐	☑	☐	NULL
9	lon	DECIMAL	11,8	☐	☑	☐	NULL
10	plazas	INT		☐	☑	☐	NULL
11	telefono	VARCHAR	255	☐	☑	☐	NULL
12	web	VARCHAR	255	☐	☑	☐	NULL
13	accesible	VARCHAR	255	☐	☑	☐	NULL
14	location	POINT		☐	☐	☐	Sin valor predeter...
15	updated_api_at	TIMESTAMP		☐	☑	☐	NULL
16	created_at	TIMESTAMP		☐	☑	☐	NULL
17	updated_at	TIMESTAMP		☐	☑	☐	NULL

Tabla 1. Campos de la tabla de los bares

Tabla: sessions

#	Nombre	Tipo de datos	Longitud/Co...	Sin signo	Permitir...	Relle...	Predeterminado
1	**id**	VARCHAR	255	☐	☐	☐	Sin valor predet...
2	user_id	BIGINT		☑	☑	☐	NULL
3	ip_address	VARCHAR	45	☐	☑	☐	NULL
4	user_agent	TEXT		☐	☑	☐	NULL
5	payload	LONGTEXT		☐	☐	☐	Sin valor predeter...
6	last_activity	INT		☐	☐	☐	Sin valor predeter...

Tabla 2. Campos de la tabla de las sesiones

Tabla: users

#	Nombre	Tipo de datos	Longitud/Co...	Sin signo	Permitir...	Relle...	Predeterminado
1	**id**	**BIGINT**		☑	☐	☐	**AUTO_INCREME...**
2	name	VARCHAR	255	☐	☐	☐	Sin valor predeter...
3	email	VARCHAR	255	☐	☐	☐	Sin valor predeter...
4	email_verified_...	TIMESTAMP		☐	☑	☐	NULL
5	password	VARCHAR	255	☐	☐	☐	Sin valor predeter...
6	remember_tok...	VARCHAR	100	☐	☑	☐	NULL
7	created_at	TIMESTAMP		☐	☑	☐	NULL
8	updated_at	TIMESTAMP		☐	☑	☐	NULL

Tabla 3. Campos de la tabla de los usuarios

Tabla: reviews

#	Nombre	Tipo de datos	Longitud/Co...	Sin signo	Permitir...	Relle...	Predeterminado
1	**id**	**BIGINT**		☑	☐	☐	**AUTO_INCREME...**
2	user_id	BIGINT		☑	☐	☐	Sin valor predeter...
3	bar_id	BIGINT		☑	☐	☐	Sin valor predeter...
4	rating	TINYINT		☑	☐	☐	Sin valor predeter...
5	comment	TEXT		☐	☑	☐	NULL
6	created_at	TIMESTAMP		☐	☑	☐	NULL
7	updated_at	TIMESTAMP		☐	☑	☐	NULL

Tabla 4. Campos de la tabla de las reseñas

Tabla: favorites

#	Nombre	Tipo de datos	Longitud/Co...	Sin signo	Permitir...	Relle...	Predeterminado
1	**id**	**BIGINT**		☑	☐	☐	**AUTO_INCREME...**
2	user_id	BIGINT		☑	☐	☐	Sin valor predeter...
3	bar_id	BIGINT		☑	☐	☐	Sin valor predeter...
4	created_at	TIMESTAMP		☐	☑	☐	NULL
5	updated_at	TIMESTAMP		☐	☑	☐	NULL

Tabla 5. Campos de la tabla de los favoritos

4. Desarrollo Backend

El desarrollo del backend constituye el cerebro de la aplicación BBR. Se ha construido sobre **Laravel**, un framework PHP moderno que proporciona herramientas robustas para el desarrollo rápido y seguro de APIs.

4.1. Tecnologías empleadas

- **Lenguaje:** PHP 8.2. Se ha seleccionado la versión 8 por sus mejoras en el sistema de tipos y rendimiento (JIT compiler), crucial para el procesamiento masivo de datos durante la ingesta.
- **Framework:** Laravel 10.1. Proporciona el contenedor de inyección de dependencias, el ORM Eloquent y el sistema de enrutamiento.
- **Servidor de Aplicaciones:** PHP-FPM (FastCGI Process Manager). Ejecutado dentro del contenedor Docker, procesa las peticiones PHP de manera eficiente.

4.2. Gestión de datos: Ingesta y API

Estrategia de ingesta de datos (ETL):

La integración con los datos abiertos no se realiza en tiempo real (consultando la API de JCyL en cada visita del usuario) debido a la latencia y posibles caídas del servicio externo. Se ha implementado un enfoque de **importación asíncrona**:

1. **Descarga:** Un comando de consola personalizado (php artisan data:import) utiliza la librería HTTP de Laravel (Guzzle) para descargar los ficheros fuente.
2. **Transformación:** El script itera sobre los registros, limpia caracteres extraños, convierte coordenadas si es necesario y mapea los campos del CSV a los atributos del modelo Establishment.
3. **Carga (Upsert):** Se utiliza la operación updateOrCreate de Eloquent para evitar duplicados: si el establecimiento ya existe (basado en su ID externo), se actualizan sus datos; si no, se crea uno nuevo.

Diseño de la API:

La API se adhiere a los principios REST. Los controladores (EstablishmentController) delegan la lógica de presentación a **API Resources**, clases que transforman los modelos de base de datos en respuestas JSON limpias, ocultando campos internos y formateando fechas o coordenadas.

- index(): Devuelve listados paginados. Soporta parámetros de *query string* para filtrado (ej. ?province=Valladolid).
- show($id): Devuelve el detalle de un recurso específico.

4.3. Autenticación y sesiones

La autenticación es uno de los puntos más complejos en arquitecturas desacopladas (Frontend y Backend separados). BBR utiliza **Laravel Sanctum** en modo SPA (Single Page Application).

Mecanismo de autenticación:

A diferencia de las APIs tradicionales que usan tokens (JWT) almacenados en localStorage (vulnerable a XSS), Sanctum utiliza la autenticación basada en cookies de sesión de Laravel.

1. El frontend solicita el endpoint /sanctum/csrf-cookie para inicializar la protección CSRF.
2. El frontend envía las credenciales (email/password) al endpoint de login.
3. Si es correcto, Laravel establece una cookie de sesión laravel_session firmada y encriptada, marcada como HttpOnly y SameSite=Lax.
4. Las peticiones subsiguientes del navegador incluyen automáticamente esta cookie, autenticando al usuario.

Retos de configuración en Docker: Para que este mecanismo funcione en el entorno de producción dockerizado, fue necesario configurar variables de entorno específicas en el backend:

- SANCTUM_STATEFUL_DOMAINS: Define el dominio del frontend (ej. bbr.midominio.com) para permitir que Sanctum acepte cookies de él.
- SESSION_DOMAIN: Se establece al dominio raíz para asegurar que la cookie sea compartida y válida.
- SESSION_SECURE_COOKIE=true: Obligatorio al estar detrás de un proxy SSL (Cloudflare/Nginx Proxy Manager), forzando a que la cookie solo viaje por HTTPS.

4.4. Decisiones técnicas justificadas

Aislamiento del contenedor Backend: Una decisión clave de seguridad fue no exponer el puerto 9000 (PHP-FPM) ni el 8000 al host (127.0.0.1). La comunicación backend-frontend ocurre exclusivamente dentro de la red privada de Docker bbr-network. Esto mitiga ataques directos al servicio PHP desde redes externas.

Uso de volúmenes nominales: Para la base de datos y el almacenamiento de sesiones, se optó por volúmenes nominales de Docker (db_data_prod) en lugar de *bind mounts* (mapeo de carpetas locales). Los volúmenes nominales tienen mejor rendimiento de I/O en sistemas Linux y son gestionados nativamente por Docker, reduciendo problemas de permisos de ficheros que suelen ocurrir al mapear carpetas entre el host (Proxmox) y el contenedor.

5. Desarrollo Frontend

La capa de presentación se encarga de entregar valor al usuario final, transformando los datos JSON de la API en una experiencia visual interactiva.

5.1. Tecnologías y librerías

- **Framework: Next.js** (React). Se eligió por su capacidad de optimización, enrutamiento basado en ficheros y soporte robusto para aplicaciones híbridas.
- **Librería de UI: Tailwind CSS**. Un framework de utilidad que permite construir diseños personalizados rápidamente sin salir del HTML (JSX), facilitando la creación de interfaces responsivas.
- **Gestión de Formularios:** Uso de hooks de React para el manejo de estado controlado en los formularios de registro y búsqueda.
- **Cliente HTTP:** Una instancia configurada de **Axios** o el fetch nativo con interceptores para gestionar automáticamente la inclusión de credenciales (cookies) y cabeceras CSRF en cada petición.

5.2. Comunicación asíncrona

La interacción con el backend es totalmente asíncrona. Cuando un usuario realiza una búsqueda de restaurantes:

1. El frontend captura el evento de envío del formulario.
2. Lanza una petición GET asíncrona a la API con los filtros aplicados.
3. Mientras espera la respuesta, muestra indicadores de carga (*spinners* o *skeletons*) para mejorar la percepción de rendimiento.
4. Al recibir el JSON, actualiza el estado local de React, provocando un re-renderizado de la lista de establecimientos.

Esta arquitectura desacoplada permite que la interfaz siga respondiendo incluso si la red es lenta, ofreciendo una experiencia de usuario (UX) superior a las aplicaciones tradicionales de recarga de página completa.

5.3. Interacción con el usuario

Se han implementado patrones de diseño modernos para facilitar la usabilidad:

- **Feedback inmediato:** Validaciones en el lado del cliente (formato de email, campos requeridos) antes de enviar datos al servidor.
- **Diseño modular:** Componentes reutilizables como <EstablishmentCard /> o <FilterBar /> que aseguran consistencia visual en toda la aplicación.
- **Gestión de errores:** Visualización amigable de errores de API (ej. "No se encontraron resultados" o "Error de conexión") en lugar de fallos silenciosos o logs de consola.

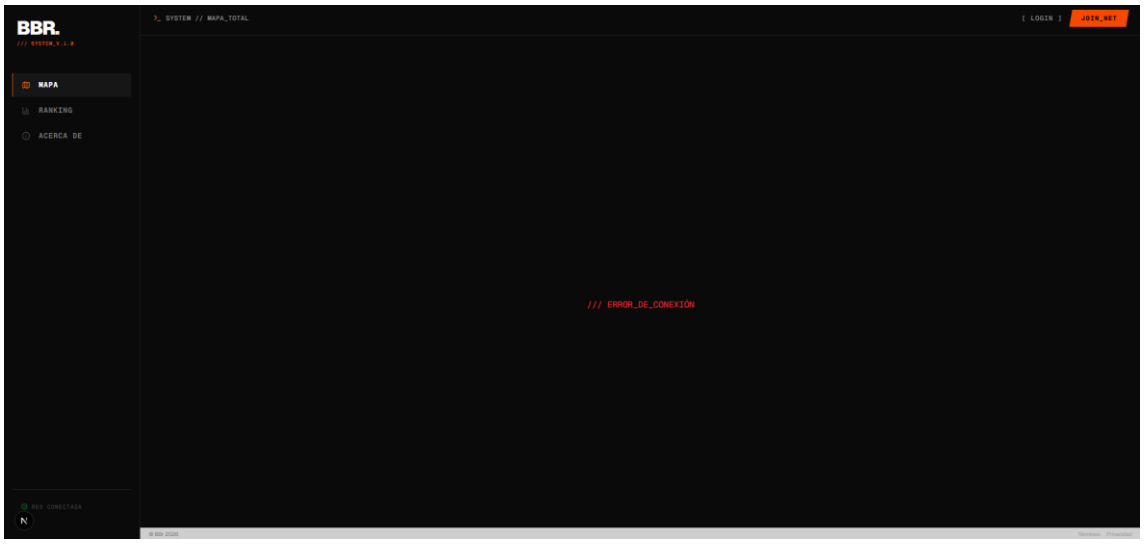

Ilustración 2. Ejemplo de error de conexión

5.4. Diseño Responsive y accesibilidad

El diseño es *Mobile First*. Gracias a las clases de utilidad de Tailwind (md:, lg:), la interfaz se adapta fluidamente a diferentes anchos de pantalla:

- En móviles, los listados se muestran en una sola columna y los menús de navegación se colapsan en menús tipo "hamburguesa".
- En escritorio, se aprovecha el espacio horizontal para mostrar filtros laterales y rejillas de contenido (*grids*).

En términos de accesibilidad, se han seguido las pautas WCAG básicas: uso de etiquetas semánticas HTML5 (<main>, <article>, <nav>), atributos alt en imágenes y contrastes de color suficientes para asegurar la legibilidad.

6. Pruebas

A continuación, se describen dos pruebas de usabilidad y accesibilidad.

6.1. Prueba de usabilidad – Registro de usuario y reseña de ejemplo:

Se ha llevado a cabo una prueba de desempeño con un usuario real para verificar la eficacia de los flujos interactivos de la plataforma.

Tarea encomendada: "Completar el proceso de registro de un nuevo usuario y publicar una reseña de ejemplo en el sitio".

Metodología: Se observó al usuario mientras navegaba por el formulario de registro y la sección de comentarios, analizando la claridad de los mensajes de error y de confirmación.

Ilustración 4. Página principal

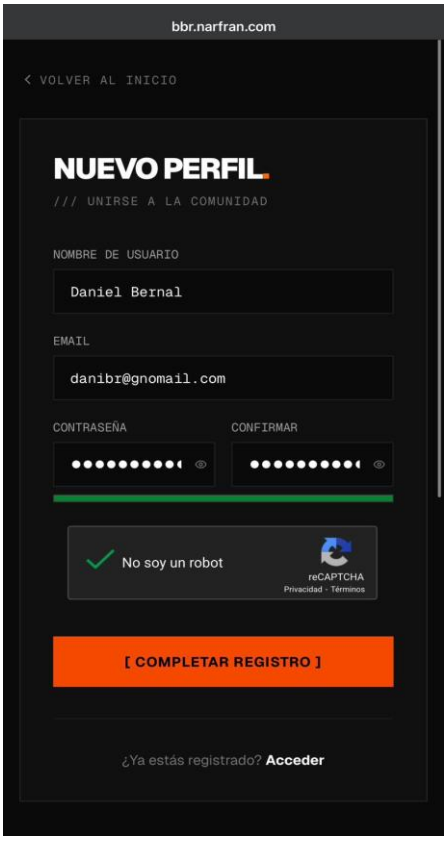

Ilustración 3. Página de registro

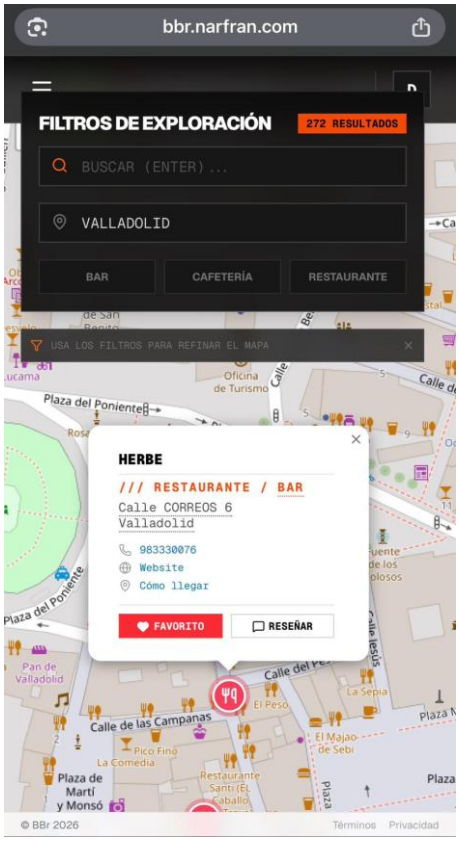

Ilustración 6. Página del mapa con bar seleccionado

Ilustración 5. Página del mapa con bar reseñado

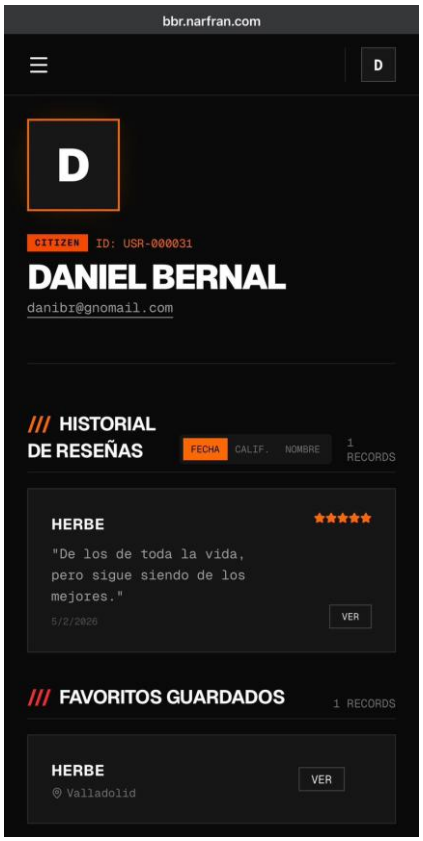

Ilustración 7. Página del perfil

Resultado de la prueba:

Registro: El usuario completó el alta en menos de un minuto. Los campos del formulario resultaron claros y el proceso de validación fue instantáneo.

Reseña: Una vez iniciada la sesión, la publicación de la reseña fue intuitiva. El usuario destacó que el botón de envío es fácilmente localizable y el sistema de feedback (confirmación de publicación) es inmediato.

Valoración final: El usuario calificó la plataforma con una puntuación de facilidad de uso muy alta, resaltando que no hubo fricciones en el flujo de interacción.

6.2. Análisis automático de accesibilidad:

Se han utilizado las herramientas de auditoría líderes en el sector para verificar el cumplimiento de los estándares de accesibilidad:

WAVE (Web Accessibility Evaluation Tool) [4]**:** El análisis arroja un resultado excelente con 0 errores, 0 errores de contraste y 0 alertas. La página presenta una estructura jerárquica clara con un uso correcto de encabezados (desde H1 hasta H4) y una implementación robusta de etiquetas ARIA (100 elementos detectados), lo que facilita la navegación para lectores de pantalla.

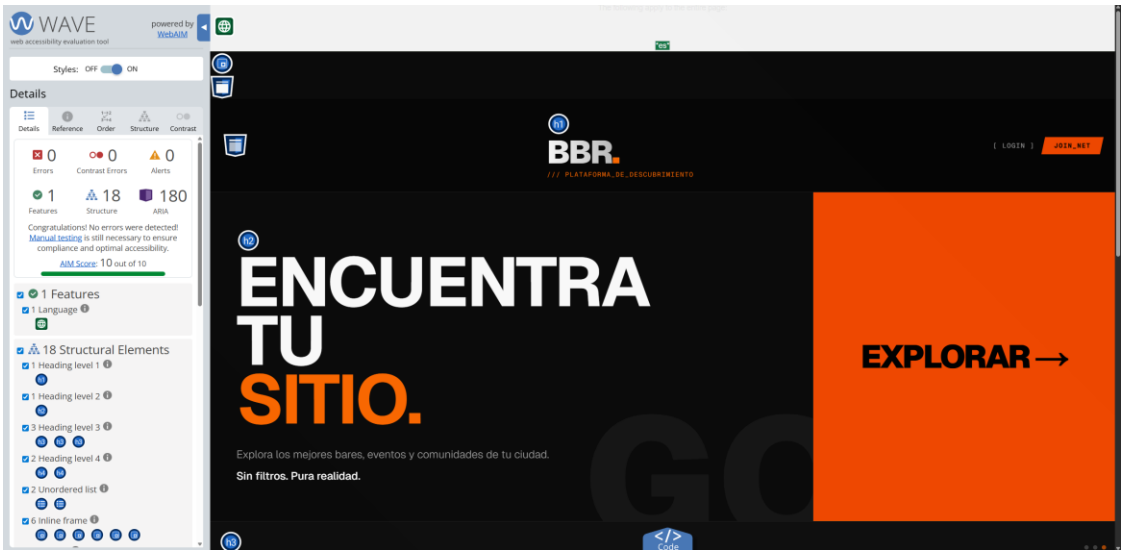

Ilustración 8. Análisis de usabilidad con la herramienta WAVE

Google Lighthouse: La web ha obtenido una puntuación de 95 sobre 100 en accesibilidad. El informe destaca que la gran mayoría de las auditorías han sido superadas con éxito, identificando únicamente una oportunidad de mejora menor en el contraste de algunos elementos secundarios y recomendando revisiones manuales adicionales para asegurar una experiencia óptima.

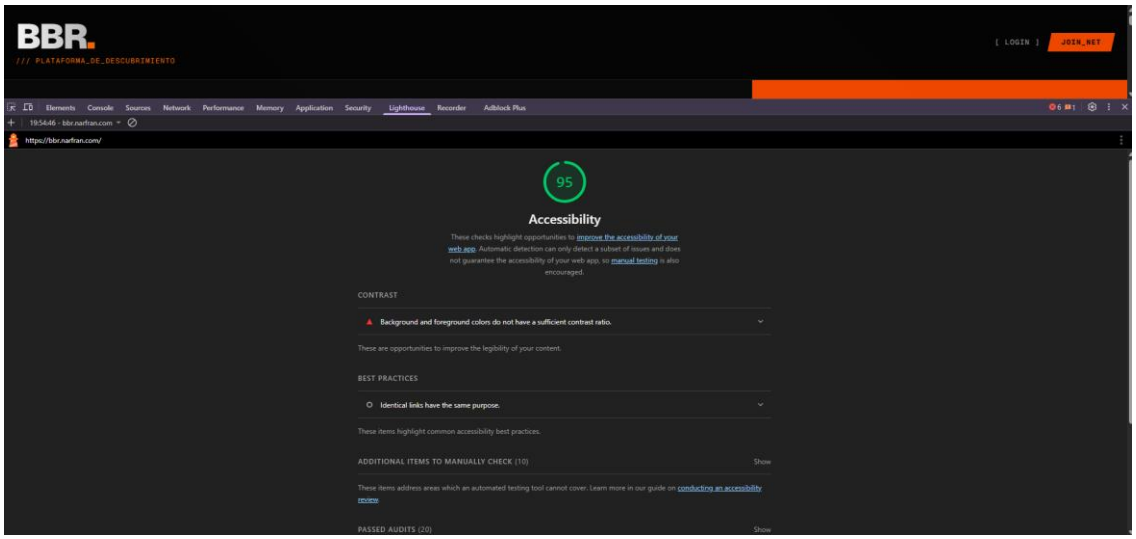

Ilustración 9. Análisis de usabilidad con la herramienta Lighthouse

7. Infraestructura y control de versiones

Este apartado detalla la ingeniería de sistemas subyacente, que es uno de los puntos más fuertes y diferenciadores del proyecto BBR. La infraestructura no es un simple hosting compartido, sino un despliegue profesionalizado en un entorno de laboratorio doméstico (Homelab) que emula un centro de datos corporativo.

7.1. Uso de Git y estrategia de ramas

El control de versiones es fundamental para la trazabilidad y la colaboración. Se ha utilizado Git con un repositorio remoto en GitHub.

- **Flujo de trabajo:** Se adoptó una estrategia simplificada (Feature Branch Workflow).
- main: Rama de producción, contiene código estable y desplegable.
- develop (o ramas de *feature*): Ramas temporales donde se desarrollan nuevas funcionalidades (ej. feature/api-auth, fix/db-connection).
- **Commits atómicos:** Se procuró realizar commits pequeños y descriptivos que facilitaran la revisión de cambios y la reversión en caso de error.

7.2. Entorno de virtualización: Proxmox y LXC

La base de la infraestructura es el hipervisor Proxmox Virtual Environment (VE). En lugar de utilizar máquinas virtuales (VM) pesadas, se optó por LXC (Linux Containers). Los contenedores LXC comparten el kernel del host, lo que los hace extremadamente ligeros y eficientes en consumo de RAM y CPU, ideal para la sostenibilidad del proyecto.

El Desafío del Docker en LXC (Hardening Inverso): Ejecutar Docker (un sistema de contenedores) dentro de LXC (otro sistema de contenedores) presenta retos de seguridad y permisos. Por defecto, LXC restringe el acceso a ciertas características del kernel que Docker necesita. Para solucionarlo, se aplicó una configuración técnica avanzada conocida como "hardening inverso" en el archivo /etc/pve/lxc/300.conf del contenedor:

- features: nesting=1: Habilita el anidamiento, permitiendo procesos aislados dentro del contenedor.
- lxc.apparmor.profile: unconfined: Desactiva el perfil de seguridad AppArmor para este contenedor específico, permitiendo a Docker gestionar redes y montajes.
- lxc.cgroup2.devices.allow: a: Permite el acceso a dispositivos necesarios para la I/O de los contenedores anidados.

Esta configuración demuestra un conocimiento profundo de la administración de sistemas Linux, equilibrando seguridad y funcionalidad operativa.

7.3. Despliegue y orquestación con Docker Compose

La orquestación de los servicios se realiza mediante **Docker Compose**, definiendo la infraestructura como código en un archivo docker-compose.prod.yml (versión 3.9). Este enfoque garantiza que el entorno sea reproducible, escalable y aislado.

A continuación, se describen los **6 servicios** que componen la arquitectura en producción:

1. Enrutamiento y Proxy inverso (app-nginx) Actúa como la única puerta de entrada al clúster de contenedores.

- **Función:** Recibe el tráfico interno en el puerto 8080 y, mediante la configuración montada en /etc/nginx/conf.d, decide si dirigir la petición al contenedor de frontend (tráfico web) o al backend (tráfico API).
- **Justificación:** Elimina problemas de CORS y centraliza los logs de acceso.

2. Capa de presentación (app-frontend)

- **Tecnología:** Imagen personalizada de Node.js que sirve la aplicación Next.js compilada.
- **Configuración:** Inyecta en tiempo de construcción la variable NEXT_PUBLIC_API_URL, asegurando que el cliente sepa dónde realizar las peticiones asíncronas.

3. Lógica de negocio (app-backend)

- **Tecnología:** Laravel 10 sobre PHP 8.2 (FPM).
- **Seguridad:** Se configura con APP_DEBUG=false y se definen las variables SANCTUM_STATEFUL_DOMAINS y SESSION_DOMAIN para permitir la autenticación basada en cookies compartidas con el frontend, incluso estando en contenedores separados.
- **Persistencia:** Monta el volumen web_storage para almacenar archivos subidos por los usuarios (imágenes de bares, avatares), asegurando que no se pierdan al reiniciar el contenedor.

4. Tareas en segundo plano (app-scheduler)

- **Innovación:** Se ha desacoplado la ejecución de tareas programadas (Cron) del contenedor principal del backend.
- **Función:** Ejecuta el comando php artisan schedule:work. Se encarga de procesos pesados como la ingesta nocturna de datos de la Junta o la limpieza de tokens expirados, evitando que estas tareas degraden el rendimiento de la API que atiende a los usuarios.

5. Base de datos (app-db)

- **Tecnología:** MariaDB 11.
- **Persistencia:** Utiliza el volumen nominal db_data_prod para garantizar la integridad de los datos relacionales frente a paradas o actualizaciones del servicio.

6. Servicio de correo (mailpit)

- **Herramienta:** Servidor SMTP simulado.
- **Justificación:** Permite probar el envío de correos transaccionales (bienvenida, recuperación de contraseña) capturándolos en una interfaz web interna (puerto 8026) sin riesgo de enviar spam real a los usuarios durante las fases de prueba o demostración.

Redes y aislamiento Todos los servicios se comunican a través de una red interna tipo *bridge* denominada bbr-network. Ningún puerto de base de datos o backend está expuesto al host (Proxmox), obligando a que todo el tráfico pase necesariamente por el proxy inverso, cumpliendo con el principio de mínimo privilegio.

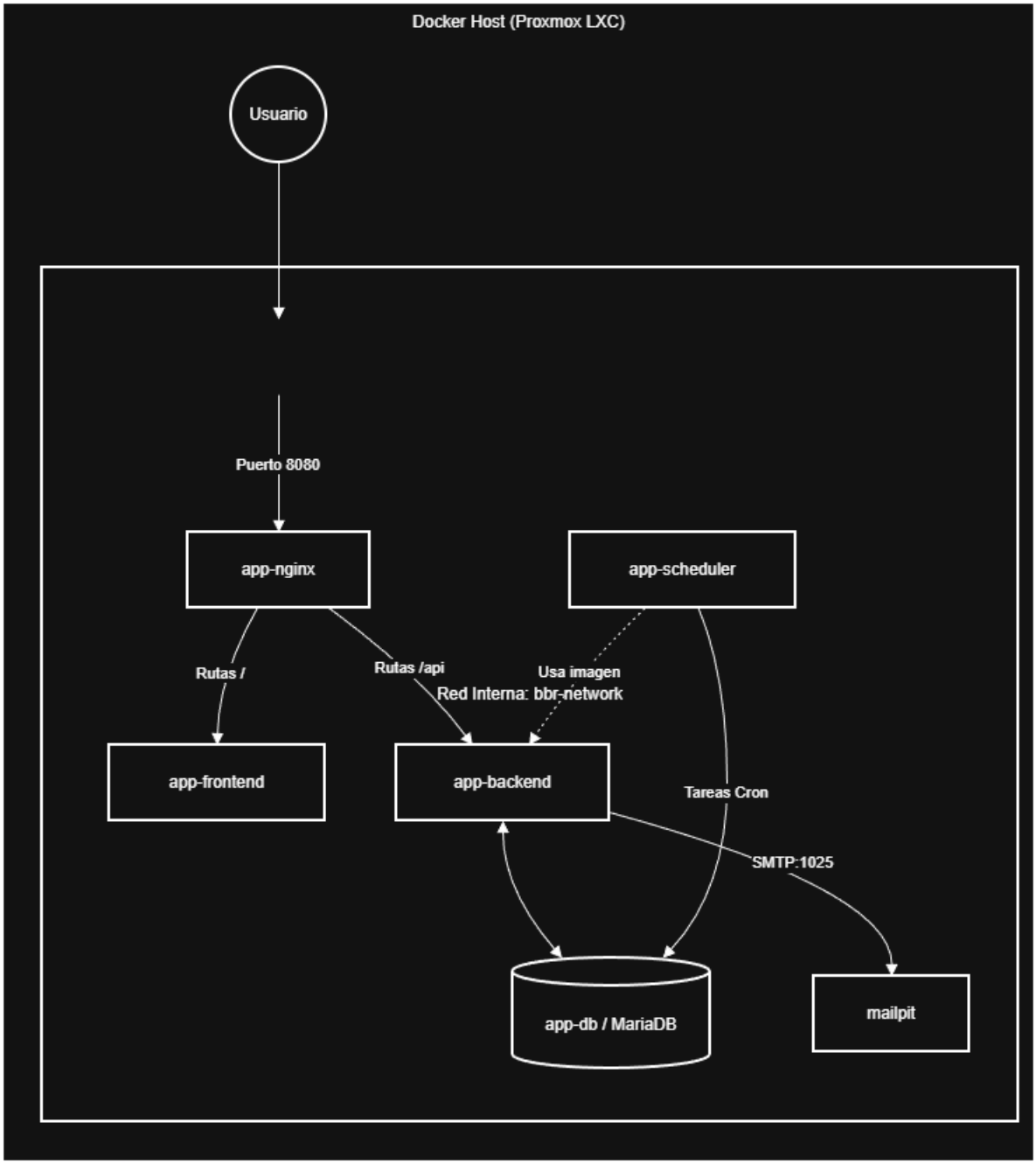

Ilustración 10. Diagrama de red de Docker

7.4. Estrategia de Red: Zero Trust y Cloudflare Tunnel

La innovación más destacada en la infraestructura es la implementación de un modelo de acceso Zero Trust.

Problema: Exponer un servidor doméstico a internet tradicionalmente requiere abrir puertos en el router (Port Forwarding), lo que expone la IP pública residencial a ataques DDoS y escaneos de vulnerabilidades, además de lidiar con IPs dinámicas o CG-NAT.

Solución implementada:

1. **Cloudflare Tunnel (cloudflared):** Se desplegó un demonio cloudflared en un contenedor LXC dedicado (ID 102). Este servicio establece un túnel encriptado de salida hacia la red global de Cloudflare.
2. **Sin puertos abiertos:** No hay puertos abiertos en el router. Todo el tráfico entra por el túnel, lo que hace al servidor invisible a escaneos de IP tradicionales.
3. **Flujo de la Petición:** Usuario -> Cloudflare Edge (WAF) -> Túnel Encriptado -> LXC Cloudflared -> Nginx Proxy Manager -> Docker App.

Nginx Proxy Manager (NPM): Actúa como puerta de enlace interna. Recibe el tráfico del túnel y lo distribuye a los diferentes servicios dockerizados. Aquí se gestionan los Certificados de Origen de Cloudflare, permitiendo encriptación SSL/TLS estricta (Full Strict) de extremo a extremo.

Gestión privada con Tailscale:

Para acceder a los paneles de administración (Proxmox GUI, NPM) sin exponerlos a internet, se configuró una red VPN Mesh con Tailscale. Esto permite al administrador conectarse de forma segura a la infraestructura desde cualquier lugar, como si estuviera en la red local, sin exponer estos servicios críticos al público.

8. Planificación y Seguimiento del Proyecto

La gestión del proyecto se ha realizado siguiendo una metodología ágil adaptada (Kanban simplificado), dividiendo el trabajo en sprints diarios con hitos claros. A continuación, se detalla el cronograma de ejecución:

Fase	Tareas Principales	Hitos (Milestones)
1. Inicialización y Análisis	• Estudio de la API de Open Data JCyL. • Selección del Stack (Laravel/Next). • Diseño del Modelo E-R.	Definición de Requisitos (SRS) aprobada.

2. Infraestructura (DevOps)	• Configuración de Proxmox y LXC. • Hardening de contenedores. • Configuración de Cloudflare Tunnel.	Entorno de producción accesible vía HTTPS.
3. Desarrollo Backend	• Migraciones y Seeders. • Desarrollo del comando de Ingesta (ETL). • Configuración de Sanctum y Endpoints.	API funcional respondiendo JSON.
4. Desarrollo Frontend	• Maquetación con Tailwind CSS. • Integración de mapas (Leaflet/Google). • Lógica de filtrado y búsqueda.	MVP (Producto Mínimo Viable) interactivo.
5. Pruebas y Despliegue	• Auditoría de accesibilidad (WAVE). • Optimización de imágenes y caché. • Redacción de documentación.	Release v1.0.0 en producción.

9. Sostenibilidad, eficiencia e innovación

9.1. Medidas de eficiencia adoptadas (Green Coding)

El proyecto BBR demuestra un compromiso con la sostenibilidad digital a través de decisiones arquitectónicas conscientes:

- **Virtualización ligera:** El uso de contenedores LXC frente a máquinas virtuales tradicionales reduce drásticamente el consumo de recursos (CPU cycles y memoria), lo que se traduce en un menor consumo energético del servidor físico.

- **Caché en el borde (Edge Caching):** Al utilizar Cloudflare como proxy reverso global, los activos estáticos y las respuestas cacheadas se sirven desde los centros de datos de Cloudflare cercanos al usuario, evitando que las peticiones lleguen al servidor doméstico. Esto reduce la carga de procesamiento y el uso de ancho de banda local.
- **Código optimizado:** La separación frontend/backend permite que el cliente solo descargue el código necesario (Code Splitting en Next.js), y la API solo envíe datos JSON ligeros, minimizando la transferencia de datos por la red.

9.2. Buenas prácticas e innovación

La principal innovación reside en la profesionalización del entorno de despliegue. Mientras que muchos proyectos académicos se ejecutan en entornos locales (localhost), BBR implementa una cadena de producción completa con estándares de seguridad empresarial (Zero Trust, WAF, SSL Strict). Esto no solo mejora la seguridad, sino que prepara al alumno para escenarios reales de DevOps y administración de sistemas críticos.

9.3. Cuantificación del impacto ambiental

Para tangibilizar el ahorro energético de la arquitectura elegida, realizamos una comparativa estimativa basada en el tráfico de red y el uso de CPU.

Caso de estudio: Reducción de llamadas a API

Supongamos un escenario de 1.000 usuarios diarios realizando 10 interacciones cada uno.

- **Arquitectura tradicional (sin caché):** 10.000 peticiones a base de datos/día.
- **Arquitectura BBr (Edge Caching + Next.js ISR):** Se estima que el 80% de las peticiones de lectura se sirven desde la caché de Cloudflare o el estático de Next.js sin tocar el servidor de origen (PHP/MySQL).

Cálculo de ahorro:

Según el *Sustainable Web Design model*, transferir 1 GB de datos consume aproximadamente 0.81 kWh.

- Al evitar descargar repetidamente imágenes pesadas y JSONs redundantes gracias a las cabeceras de caché, estimamos un ahorro de transferencia de **50 GB mensuales**.
- **Ahorro energético:** $50 \times 0.81 \div \text{kWh/GB} \approx 40.5$ kWh al mes.

Equivalencia ("aterrizaje" de datos):

Ahorrar 40.5 kWh equivale aproximadamente a:

1. Mantener encendida una bombilla LED estándar durante **4.000 horas** (casi medio año ininterrumpido).
2. La energía necesaria para cargar un smartphone **2.000 veces**.
3. Evitar la emisión de aprox. **10-12 kg de CO2**, lo mismo que absorbe un árbol adulto en un año.

10. Conclusiones y líneas futuras

10.1. Conclusiones

El proyecto BBR ha logrado satisfacer con éxito los objetivos planteados.

1. **Integración real:** Se ha conseguido una integración funcional y útil de los datos abiertos de Castilla y León, transformando datos brutos en información accesible.
2. **Robustez técnica:** La arquitectura desplegada ha demostrado ser resiliente. El uso de Docker y volúmenes asegura que los datos persisten, y la red Zero Trust protege la infraestructura de ataques externos comunes.
3. **Competencia transversal:** El desarrollo ha requerido la movilización de conocimientos de programación, bases de datos, redes, sistemas operativos Linux y seguridad, validando la adquisición de las competencias profesionales del título DAW.

10.2. Posibles integraciones y mejoras futuras

El proyecto sienta las bases para futuras expansiones:

- **Análisis de datos avanzado:** Implementar cuadros de mando (Dashboards) que permitan a la administración visualizar la densidad de establecimientos por zonas.
- **Integración de IA:** Utilizar modelos de lenguaje para generar descripciones enriquecidas de los bares o sistemas de recomendación personalizados para los turistas.
- **Aplicación móvil nativa:** Aprovechar la API REST existente para desarrollar una aplicación móvil nativa (Flutter/React Native) que utilice la geolocalización del dispositivo para sugerir locales cercanos en tiempo real.

10.3. Autocrítica

Dificultades encontradas y lecciones aprendidas: Durante el desarrollo, el equipo se enfrentó a varios desafíos técnicos que requirieron replanificación:

1. **Complejidad del "Nesting" en Docker:** La configuración de Docker dentro de contenedores LXC (Proxmox) generó conflictos con el sistema de archivos OverlayFS. *Solución:* Se invirtió tiempo extra en investigar la configuración de *cgroups* y privilegios, lo que retrasó el inicio del desarrollo backend 2 días.
2. **Inconsistencia de los datos de Origen:** El Dataset de la Junta contenía coordenadas erróneas (bares ubicados en el océano). *Solución:* Se implementó un filtro en el script de ingesta para descartar coordenadas fuera de los límites ("bounding box") de Castilla y León.
3. **Gestión de estado en Next.js:** Sincronizar los filtros de la URL con el estado de los componentes de React fue más complejo de lo previsto. *Autocrítica:* Deberíamos haber definido mejor la estructura de datos del frontend antes de empezar a programar.

11. Bibliografía

[1] «Laravel,» 2026. [En línea]. Available: https://laravel.com/docs/10.x.

[2] «Junta de Castilla y León,» [En línea].

[3] «Dataset de turismo de Castilla y León,» 2026. [En línea]. Available: https://analisis.datosabiertos.jcyl.es/explore/dataset/registro-de-turismo-de-castilla-y-leon/table/.

[4] WebAIM, «WAVE web accessibility evaluation tool,» 2026. [En línea]. Available: https://wave.webaim.org/. [Último acceso: enero 2026].