Instituto Puig Castellar
Ciclo formativo: DAM2A
Grado: CFGS
Crédito de Síntesis / Proyecto intermodular

Rediseño web de la Escola Lluís Millet

Diseño y desarrollo de un sitio web institucional moderno

Autor/a/es: Jose Ribelles García
Tutor: Luis Elía Talón
Curso: 2025-2026
Fecha de entrega: 17/05/2026

Resumen del proyecto

El presente proyecto consiste en el rediseño completo de la web de la Escola Lluís Millet, un centro educativo concertado de Barcelona, desarrollado como proyecto de final de grado superior de Desarrollo de Aplicaciones Multiplataforma (DAM) durante el curso 2025–2026. El proyecto nace de una petición real de la directora del centro y el tutor de prácticas, quienes sin conocimientos técnicos en desarrollo web encargan al alumno en prácticas diseñar y construir desde cero una nueva página web para el colegio. Partiendo de un análisis exhaustivo de la web existente y de una comparativa con cuatro centros de referencia, se elaboró una propuesta de arquitectura web, una guía de estilo institucional con paleta de colores en tonos verde oliva, tipografías (Montserrat y Open Sans) y criterios visuales, y se desarrolló el sitio con Next.js en el frontend y Strapi como CMS headless en el backend, permitiendo al equipo del centro gestionar los contenidos de forma autónoma.

Sin embargo, el proyecto también refleja fielmente la realidad del trabajo con un cliente sin perfil técnico: cambios de criterio durante el desarrollo, dificultades de comunicación con la dirección del centro y, finalmente, la decisión del centro de no contratar hosting de pago ni dominio propio por no sentirse capaz de gestionar la plataforma de forma independiente. Este desenlace llevó a desarrollar una segunda versión del sitio sobre la plataforma NODES2 (xtecagora) de la Generalitat de Catalunya, una solución más básica similar a WordPress que no requiere infraestructura propia ni coste adicional para el centro, y que convive con la versión técnica completa desarrollada en Next.js y Strapi como demostración del trabajo realizado durante las prácticas. El resultado final documenta todo ese proceso real: desde el análisis y el diseño hasta el desarrollo técnico con tecnologías profesionales y la adaptación a las limitaciones concretas del cliente.

Índice

1. Presentación del proyecto

1.1 Introducción

El presente proyecto consiste en el rediseño integral de la web de la Escola Lluís Millet, un centro educativo concertado en el barrio de les Oliveres (Barcelona) que imparte las etapas de Infantil, Primaria y Secundaria. El trabajo ha sido desarrollado en el marco de las prácticas del Ciclo Formativo de Grado Superior de Desarrollo de Aplicaciones Multiplataforma (DAM) durante el curso 2025–2026.

El punto de partida del proyecto es un encargo real: la directora del centro y el tutor de prácticas, sin conocimientos técnicos en desarrollo web, solicitan al alumno en prácticas que diseñe y construya desde cero una nueva presencia digital para el colegio. Este punto de partida es relevante porque condiciona toda la forma de trabajar a lo largo del proyecto: no existe un briefing técnico definido, los requisitos se van concretando de forma progresiva a través de reuniones, formularios y propuestas, y las decisiones de diseño y tecnología deben tomarse teniendo siempre en cuenta que el cliente final no tiene perfil técnico y necesita poder gestionar el sitio de forma autónoma una vez entregado.

El proyecto abarca todo el ciclo completo de creación de un sitio web institucional moderno. Comienza con las primeras reuniones de toma de requisitos, continúa con el análisis exhaustivo de la web existente y el estudio comparativo con cuatro centros educativos de referencia, avanza hacia la definición de la identidad visual mediante una guía de estilo institucional y el prototipado en Figma, y culmina con el desarrollo técnico completo de la plataforma utilizando Next.js con App Router en el frontend y Strapi como CMS headless en el backend.

Esta combinación tecnológica responde directamente a la necesidad del cliente: permite que el equipo del centro pueda actualizar y gestionar los contenidos del sitio de forma autónoma desde un panel de administración intuitivo, sin necesidad de intervención técnica, mientras el frontend garantiza un rendimiento óptimo, una navegación fluida y una experiencia de usuario moderna en cualquier dispositivo.

El recorrido del proyecto también incluye los imprevistos y dificultades propios de trabajar con un cliente real no técnico: cambios de criterio durante el desarrollo, dificultad para obtener feedback y validaciones por parte de la dirección del centro, y finalmente la decisión de no contratar un servicio de hosting de pago ni adquirir un dominio propio. Este desenlace llevó a desarrollar una segunda versión del sitio sobre la plataforma NODES2 (xtecagora) de la Generalitat de Catalunya, con un diseño más básico adaptado a las limitaciones reales del centro, que es la solución que el centro utilizará en producción. El proyecto presenta por tanto dos entregas diferenciadas: la solución técnica completa con Next.js y Strapi, que representa el núcleo del trabajo de análisis, diseño y desarrollo, y la adaptación funcional sobre NODES2, que es la versión que el centro puede gestionar sin infraestructura propia ni coste adicional.

El resultado final es una web institucional responsiva, accesible y visualmente coherente con la identidad del centro, acompañada de documentación técnica completa y de una versión adaptada a las posibilidades reales del cliente, que en conjunto reflejan fielmente todo el proceso real de un proyecto web profesional.

1.2 Contexto

La Escola Lluís Millet es un centro educativo concertado de Barcelona que imparte Educación Infantil, Primaria y Secundaria en el barrio de les Oliveres. Como la mayoría de instituciones educativas de su entorno, el centro utiliza su presencia web como canal principal de comunicación con las familias, para difundir información sobre las etapas educativas, publicar noticias y facilitar el acceso a documentos, recursos y trámites de secretaría.

En el momento de iniciar el proyecto, el sitio web del centro estaba alojado en la plataforma blocs.xtec.cat, el servicio de blogs educativos de la Generalitat de Catalunya basado en WordPress. Esta solución, pensada originalmente como un blog y no como un sitio web institucional, presentaba limitaciones estructurales importantes que condicionaban toda la experiencia del usuario.

Desde el punto de vista del diseño y la estructura, el sitio tenía una apariencia claramente anticuada, con una estética de blog que no transmitía la imagen profesional que el centro quería proyectar. El menú principal mezclaba secciones de muy distinta naturaleza sin una jerarquía clara: junto a apartados como "El Centre Educatiu" o "Serveis" aparecían entradas independientes como "INSTAL·LACIÓ DEL JAVA" o premios y reconocimientos al mismo nivel de navegación que contenidos esenciales para las familias. No existía una diferenciación por etapas educativas accesible desde el menú, y secciones clave como el contacto, los horarios de secretaría o la información de matrícula no tenían una ubicación clara ni destacada.

Desde el punto de vista técnico, el sitio no era responsivo y no se adaptaba correctamente a dispositivos móviles, lo que suponía una barrera significativa teniendo en cuenta que la mayoría de usuarios accede a este tipo de sitios desde el teléfono. Tampoco cumplía con los requisitos de accesibilidad del estándar WCAG 2.1, las imágenes carecían de textos alternativos, y no se incluían ni política de privacidad ni aviso legal, incumpliendo la normativa europea vigente en materia de protección de datos (RGPD). Tampoco aparecían los vínculos institucionales con la Generalitat de Catalunya requeridos para centros concertados.

Desde el punto de vista visual, la identidad gráfica del centro no estaba sistematizada en ningún documento de referencia. Esto se traducía en incoherencias tipográficas, cromáticas y de composición a lo largo del sitio, sin jerarquía visual entre títulos y contenidos y sin una línea gráfica coherente que conectase el sitio con la imagen institucional del centro.

En este contexto, el centro de prácticas propuso llevar a cabo el rediseño completo del sitio como proyecto de final de grado. La petición llegó de la mano de la directora y el tutor de prácticas, quienes tenían claro que querían una web nueva y moderna, pero no tenían definido el enfoque técnico, la estructura de contenidos ni los elementos visuales que debía incluir. Esta situación obligó a trabajar la toma de requisitos de forma activa desde el primer día, mediante reuniones, formularios de feedback y propuestas iterativas, antes de poder avanzar hacia las fases de diseño y desarrollo.

Captura de la web original de la Escola Lluís Millet en blocs.xtec.cat en el momento de iniciar el proyecto.

1.3 Justificación

La necesidad de renovar la web de la Escola Lluís Millet quedó evidenciada desde el primer momento en que se accedió al sitio existente. El análisis exhaustivo realizado en la fase previa del proyecto confirmó y documentó lo que ya era visible a simple vista: el sitio no cumplía con los estándares mínimos que se esperan de la web de un centro educativo en la actualidad, ni desde el punto de vista técnico, ni desde el comunicativo, ni desde el visual.

Desde el punto de vista del usuario, la web no ofrecía una navegación clara ni diferenciada por etapas educativas. El menú principal mezclaba secciones de naturaleza muy distinta sin jerarquía lógica, lo que dificultaba encontrar información básica. Las familias no podían acceder fácilmente a contenidos clave como horarios de secretaría, calendarios escolares o documentos de matrícula. A esto se sumaba la falta de adaptación a dispositivos móviles, una barrera especialmente relevante dado que la mayoría de usuarios accede a este tipo de sitios desde el teléfono.

Desde el punto de vista técnico, el sitio no era responsivo, las imágenes carecían de textos alternativos, y no se incluían ni política de privacidad ni aviso legal, incumpliendo directamente la normativa europea vigente en materia de protección de datos (RGPD). Tampoco se mostraban los vínculos institucionales con la Generalitat de Catalunya requeridos para centros concertados. La plataforma sobre la que estaba construido el sitio, blocs.xtec.cat, es un servicio de blogs educativos que no está pensado para funcionar como un sitio web institucional moderno, lo que imponía limitaciones estructurales difíciles de resolver sin un rediseño completo.

Desde el punto de vista visual e institucional, la web transmitía una imagen poco profesional que no reflejaba la identidad ni los valores del centro. No existía ninguna guía de estilo definida, lo que derivaba en incoherencias tipográficas, cromáticas y de composición a lo largo de todo el sitio. El logotipo original del centro, basado en un degradado dorado con detalles artesanales, perdía nitidez en entornos digitales y no estaba adaptado para uso web.

El análisis comparativo con cuatro centros educativos de referencia —FEDAC Santa Coloma, Escola Manent, CEIP Serra de Marina y St Peter's School Barcelona— confirmó que el Lluís Millet presentaba las mayores carencias entre los centros analizados, especialmente en accesibilidad y adaptabilidad móvil, y que existían modelos consolidados de los que extraer buenas prácticas aplicables al contexto del centro.

Por todo ello, el rediseño integral de la web no solo era conveniente, sino necesario para que el centro dispusiera de una presencia digital a la altura de su trayectoria pedagógica y de las expectativas de su comunidad educativa.

1.4 Objetivos

Objetivo general

Diseñar y desarrollar una nueva web institucional para la Escola Lluís Millet que sea moderna, accesible, visualmente coherente y técnicamente robusta, y que permita al equipo del centro gestionar sus contenidos de forma autónoma sin necesidad de conocimientos técnicos.

Objetivos específicos

Análisis y planificación :

- Analizar en profundidad la web actual del centro, identificando sus puntos débiles en diseño, estructura, contenidos y accesibilidad, tomando como referencia directa el sitio existente en blocs.xtec.cat.
- Comparar el sitio con cuatro centros educativos de referencia para extraer buenas prácticas y definir un modelo adecuado al contexto del Lluís Millet.
- Trabajar la toma de requisitos con un cliente no técnico mediante reuniones, formularios de feedback y propuestas iterativas, hasta llegar a una arquitectura de contenidos validada por el centro.
- Proponer y validar una nueva estructura de navegación clara, jerárquica y orientada al usuario, con tres alternativas de organigrama para que el centro pudiera elegir la que mejor se ajustara a sus necesidades.

Diseño e identidad visual :

- Elaborar una guía de estilo institucional que unifique la identidad visual del centro: logotipo, paleta de colores, tipografía, iconografía y criterios fotográficos.
- Crear un prototipo en Figma que sirva como referencia visual validada antes de iniciar el desarrollo, permitiendo obtener feedback del cliente sobre el diseño antes de escribir código.
- Diseñar una interfaz responsiva, limpia y accesible, adaptada a cualquier dispositivo y coherente con los valores del centro.

Desarrollo técnico :

- Implementar el frontend de la web con Next.js (App Router), garantizando un rendimiento óptimo y una arquitectura de componentes escalable y mantenible.
- Integrar Strapi como CMS headless para la gestión dinámica de contenidos mediante una Dynamic Zone modular, que permita al equipo del centro actualizar el sitio sin conocimientos técnicos.
- Asegurar el cumplimiento de los estándares de accesibilidad WCAG 2.1 y de la normativa legal vigente en materia de privacidad (RGPD).
- Documentar el funcionamiento del CMS mediante un manual detallado orientado a usuarios no técnicos, para que el centro pueda gestionar el sitio de forma autónoma una vez entregado.

Resultado

- Entregar un sitio web funcional, responsivo y mantenible que represente fielmente la identidad de la Escola Lluís Millet y mejore de forma significativa la comunicación con familias, alumnado y profesorado, acompañado de toda la documentación necesaria para su mantenimiento y evolución futura.

2. Estrategia y planificación

2.1 Estrategia de desarrollo y viabilidad

La estrategia adoptada para este proyecto se basa en una aproximación progresiva y por capas: primero comprender el problema antes de diseñar, primero diseñar antes de desarrollar, y validar cada fase antes de avanzar a la siguiente. Este enfoque resultó especialmente importante en este proyecto dado que el cliente no tenía perfil técnico y los requisitos no estaban definidos desde el inicio, lo que hacía imprescindible construir sobre una base sólida y validada en cada etapa antes de continuar. De este modo se reduce el riesgo de tener que rehacer trabajo y se permite incorporar el feedback real del centro de forma ordenada.

Viabilidad técnica

Las tecnologías seleccionadas son maduras, bien documentadas y ampliamente utilizadas en proyectos web profesionales. Next.js con App Router es uno de los frameworks de React más extendidos en el sector, lo que garantiza soporte a largo plazo y facilidad de mantenimiento. Strapi es el CMS headless de código abierto más popular, con una comunidad activa y una curva de aprendizaje asumible para un perfil de DAM. Figma, utilizado en la fase de prototipado, es la herramienta de diseño de interfaces más utilizada en el sector y permitió validar el diseño con el centro antes de escribir una sola línea de código.

La combinación de Next.js y Strapi es viable técnicamente porque ambas herramientas están diseñadas para trabajar juntas mediante API REST: Strapi expone los endpoints y Next.js los consume en tiempo de renderizado. La arquitectura de Dynamic Zone con módulos reutilizables permite además que el sitio crezca añadiendo nuevas secciones sin modificar el código base, lo que resulta especialmente relevante para un centro educativo que necesita poder ampliar sus contenidos en el futuro sin depender de un desarrollador para cada cambio.

El desarrollo se realizó en local durante la fase de construcción, con despliegue progresivo usando Vercel para el frontend y Render para el backend de Strapi, lo que permitió que el

tutor de prácticas y la dirección del centro pudieran ver el resultado real en cualquier momento sin necesidad de infraestructura propia.

Viabilidad temporal

El proyecto se ha desarrollado en el marco de las prácticas del grado superior, con una duración aproximada de cinco meses, desde octubre de 2025 hasta marzo de 2026. Este tiempo es suficiente para completar las fases de análisis, diseño, desarrollo y documentación siempre que se mantenga una planificación disciplinada y se prioricen correctamente las funcionalidades esenciales frente a las mejoras opcionales. En la práctica, la planificación se vio afectada en varias ocasiones por la dificultad para obtener validaciones y feedback por parte de la dirección del centro en los plazos previstos, lo que obligó a ajustar el ritmo de trabajo y a tomar determinadas decisiones de diseño sin confirmación explícita del cliente.

Viabilidad de recursos

El proyecto no requiere inversión económica significativa. Las herramientas principales utilizadas —Next.js, Strapi, Visual Studio Code, GitHub, Figma— son gratuitas o de código abierto. Las fuentes tipográficas se obtienen de Google Fonts sin coste, y las imágenes del sitio proceden del propio centro educativo, por lo que no hay dependencia de bancos de imágenes externos.

Desde el punto de vista de los recursos, las herramientas principales utilizadas —Next.js, Strapi, Visual Studio Code, GitHub, Figma— son gratuitas o de código abierto. Las fuentes tipográficas se obtienen de Google Fonts sin coste, y las imágenes del sitio proceden del propio centro educativo.

Para la puesta en producción real del proyecto se utilizaron cuatro servicios externos en sus planes gratuitos: Vercel para el despliegue del frontend, Render para el backend de Strapi, Supabase como base de datos PostgreSQL gestionada en la nube, y Cloudinary como servicio de almacenamiento y entrega de imágenes con optimización automática. La monitorización del estado del backend se configuró mediante UptimeRobot, que envía pings cada cinco minutos a la instancia de Render para evitar que el servicio entre en estado de suspensión por inactividad, problema habitual en los planes gratuitos de esta plataforma. Todos estos

servicios funcionan coordinados sin coste adicional dentro de los límites de uso del proyecto.

El único coste que el proyecto requería para una implantación definitiva era la contratación de un dominio propio. Sin embargo, durante la fase final del proyecto, la dirección del centro comunicó que no tenía intención de asumir ese coste, argumentando que no estaba segura de poder gestionar la plataforma de forma independiente. Esta decisión condicionó el desenlace del proyecto y llevó a desarrollar una segunda versión del sitio sobre la plataforma NODES2 (xtecagora) de la Generalitat de Catalunya.

2.2 Metodología de trabajo

La metodología aplicada ha sido de tipo iterativo e incremental, inspirada en los principios del desarrollo ágil, adaptada a un proyecto individual con un único desarrollador y un cliente real sin perfil técnico. Esta adaptación fue necesaria porque la naturaleza del cliente condicionó directamente la forma de trabajar: al no tener los requisitos definidos desde el inicio, no era posible planificar un desarrollo lineal tradicional, y el ciclo de análisis, propuesta, validación y ajuste tuvo que repetirse en varias ocasiones a lo largo de todo el proyecto.

Trabajo por fases con entregas incrementales

El proyecto se dividió en fases claramente diferenciadas: toma de requisitos y análisis, guía de estilo y prototipado en Figma, configuración del entorno técnico, desarrollo de módulos y contenidos, y ajustes finales. Dentro de la fase de desarrollo se trabajó de forma especialmente iterativa: en lugar de construir toda la web de golpe, se fueron desarrollando los módulos de la Dynamic Zone uno a uno, verificando el correcto funcionamiento de cada uno antes de continuar con el siguiente. Esto permitió detectar y corregir errores de forma temprana sin que afectaran al resto del sistema, y facilitó mostrar avances parciales al tutor y al centro en cualquier momento.

Revisión continua con el tutor y el centro

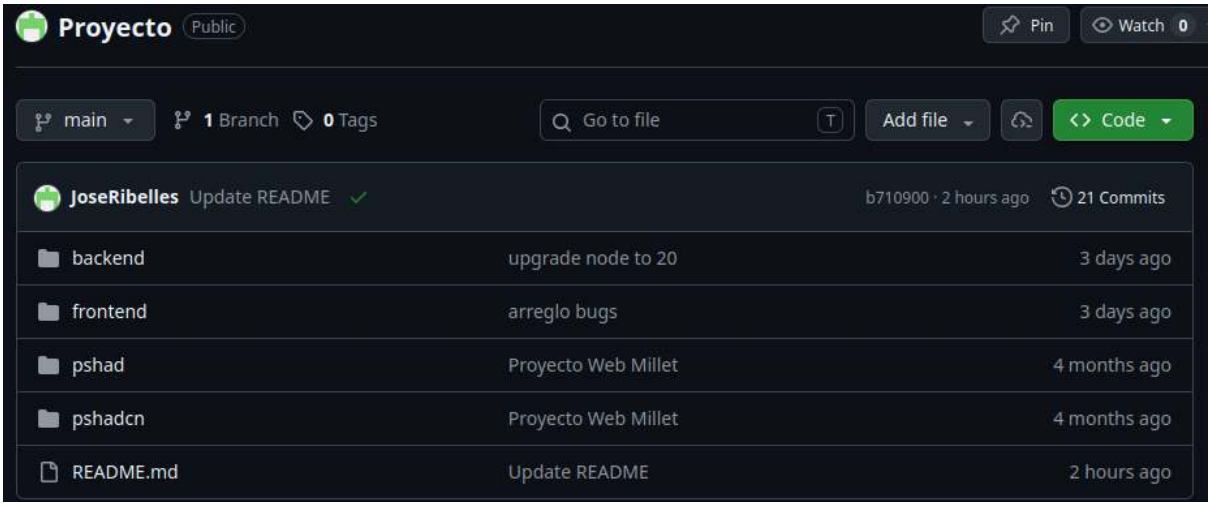
A lo largo del proyecto se realizaron revisiones periódicas con el tutor de proyecto Luis Elía y con la dirección del centro, especialmente en los hitos clave: validación del análisis comparativo, aprobación de la guía de estilo, revisión del prototipo Figma y valoración del resultado del desarrollo. Sin embargo, obtener validaciones por parte de la dirección del centro resultó más difícil de lo esperado en la práctica. En varias ocasiones la directora no acudió a las reuniones acordadas ni revisó los avances en los plazos establecidos, lo que obligó a tomar algunas decisiones de diseño sin confirmación explícita y a ajustar la planificación sobre la marcha. Este aspecto, aunque supuso una dificultad real, también forma parte del aprendizaje del proyecto: trabajar con un cliente que no siempre está disponible es una situación habitual en el entorno profesional.

Gestión de versiones con Git

El código del proyecto se gestionó mediante Git con repositorio en GitHub, lo que permitió llevar un control claro de los cambios, revertir modificaciones problemáticas y mantener un historial del progreso del desarrollo. Se trabajó con una rama principal estable y ramas de trabajo para cada módulo o funcionalidad nueva, garantizando que el código en producción siempre se encontrara en un estado funcional.

Priorización de funcionalidades

Dado que el tiempo disponible era limitado y la obtención de feedback del cliente no siempre fue fluida, se aplicó una priorización basada en el valor para el usuario final. Las funcionalidades esenciales —estructura de páginas, módulos principales de la Dynamic Zone, gestión de contenidos desde Strapi y diseño responsivo— se desarrollaron primero. Las mejoras opcionales como animaciones, integraciones avanzadas u optimizaciones de rendimiento adicionales se abordaron únicamente cuando el núcleo del proyecto estaba completamente estable. Esta priorización resultó clave para poder entregar un producto funcional y completo dentro del plazo de las prácticas, a pesar de los imprevistos surgidos durante el desarrollo.



Vista del repositorio GitHub del proyecto con las carpetas principales del sistema.

2.3 Planificación

El proyecto se estructuró en cinco fases principales distribuidas a lo largo de los cinco meses de duración del trabajo, desde octubre de 2025 hasta marzo de 2026. La planificación siguió el enfoque progresivo descrito en la metodología: no se avanzaba a la siguiente fase hasta haber completado y validado la anterior, aunque en la práctica algunos solapamientos fueron inevitables debido a los retrasos en la obtención de feedback por parte del centro.

Fase 0 — Prototipo inicial y toma de requisitos (octubre 2025)

Antes de iniciar el análisis formal, las primeras semanas de prácticas se dedicaron a las reuniones iniciales con la directora del centro y el tutor de prácticas para entender qué querían y cómo lo querían. En esta fase quedó claro que el cliente no tenía definido ni el enfoque técnico ni la estructura de contenidos, por lo que se elaboró un formulario de feedback que se envió a la dirección para recopilar sus preferencias y objetivos de forma estructurada. Paralelamente se comenzó a trabajar en un primer prototipo en Figma como punto de partida visual, que sirvió para explorar la distribución de los elementos, la jerarquía de la información y la identidad estética del sitio antes de comprometerse con ninguna decisión técnica.

Fase 1 — Análisis (noviembre 2025)

Con los requisitos iniciales recogidos, se realizó el estudio exhaustivo de la web existente del centro en blocs.xtec.cat, identificando sus problemas de diseño, accesibilidad, estructura y contenidos. Este análisis se complementó con un estudio comparativo de cuatro centros educativos de referencia: FEDAC Santa Coloma, Escola Manent, CEIP Serra de Marina y St Peter's School Barcelona. El trabajo de esta fase quedó recogido en el documento "Anàlisi Lluís Millet". A partir de las conclusiones del análisis y de las respuestas al formulario enviado al centro, se elaboró el documento "Proposta Organigrama Lluís Millet", que recogía tres alternativas de arquitectura de navegación para que la dirección pudiera elegir la estructura definitiva de la nueva web.

Fase 2 — Diseño e identidad visual (diciembre 2025)

Con el organigrama validado, se definió la guía de estilo institucional del centro: propuesta de evolución del logotipo hacia un diseño vectorial moderno, paleta de colores basada en tonos verde oliva, tipografías Montserrat y Open Sans, criterios de iconografía y fotografía, y diseño de los componentes de interfaz principales (botones, formularios, tarjetas, cabeceras). Todo ello quedó recogido en el documento "Guía de Estilo", que sirvió como referencia visual para todo el desarrollo posterior y garantizó la coherencia gráfica del sitio desde el primer momento.

Fase 3 — Configuración del entorno y arquitectura (diciembre 2025 – enero 2026)

Con el diseño definido, se configuró el entorno técnico completo: proyecto Next.js con App Router, instalación y configuración de Strapi como CMS headless, definición de los content-types en Strapi (Page, Dynamic Zone y los primeros módulos), e implementación del sistema de enrutamiento dinámico y del componente ModuleRenderer. Esta fase fue especialmente intensa en cuanto a toma de decisiones arquitectónicas, ya que la estructura definida aquí condicionaría todo el desarrollo posterior. Para documentar el funcionamiento del sistema y facilitar su uso por parte del equipo del centro, se elaboró el documento "Document d'explicació de funcionament de Strapi", un manual completo orientado a usuarios sin conocimientos técnicos.

Fase 4 — Desarrollo de módulos y contenido (enero – febrero 2026)

Con la arquitectura en pie, se desarrollaron de forma iterativa todos los módulos de la Dynamic Zone: HeroSection, HeroAvanzado, SecciónTexto, CajaDestacada, TarjetasGrid, CTASection, Acordeón, PestañasContenido, Galería, MapaUbicación, VideoIncrustado, LíneaTiempo y CuadrículaEstadísticas. Para cada módulo se siguió el mismo proceso: definición del content-type en Strapi, implementación del componente React, integración en ModuleRenderer, carga de contenidos reales del centro y revisión visual. Esta fase también incluyó el despliegue del sitio en Vercel y Render para que el centro pudiera verlo en funcionamiento real, así como la resolución de los errores de populate y los ajustes necesarios en la API de Strapi.

Fase 5 — Ajustes, revisión y documentación (marzo 2026)

La fase final se dedicó a la revisión completa del sitio: comprobación de accesibilidad y responsividad en distintos dispositivos, correcciones de diseño y contenido tras la validación con el centro, y gestión del cambio de criterio respecto al hosting y el dominio, que derivó hacia la plataforma NODES2 (xtecagora) como solución de despliegue definitiva. Paralelamente se redactó la memoria del proyecto y se preparó la presentación final.

3. Análisis

3.1 Casos de uso

Los casos de uso describen las interacciones principales entre los distintos tipos de usuario y el sistema. En este proyecto se identifican tres perfiles de usuario: el visitante público (familias, alumnado potencial, ciudadanía en general), el usuario interno del centro (dirección, secretaría, docentes) y el administrador técnico (desarrollador o responsable del mantenimiento del sistema). La definición de estos perfiles fue especialmente importante en este proyecto dado que uno de los requisitos fundamentales del cliente era que el sistema fuera gestionable de forma autónoma por personas sin conocimientos técnicos, lo que condicionó directamente las decisiones de arquitectura y la elección del CMS.

CU-01 — Consultar información del centro

- **Actor:** Visitante público (familias, alumnado potencial, ciudadanía en general)
- **Descripción:** El usuario accede a la web y navega por las diferentes secciones para obtener información sobre el centro, sus etapas educativas, metodología, proyectos y vida escolar.
- **Flujo:** El usuario entra en la página de inicio → navega por el menú principal → accede a la sección deseada → lee el contenido.
- **Resultado:** El usuario obtiene la información que buscaba de forma clara, estructurada y sin necesidad de registro ni pasos adicionales.

CU-02 — Consultar información de contacto y ubicación

- **Actor:** Visitante público
- **Descripción:** El usuario necesita saber cómo llegar al centro, conocer los horarios de atención de secretaría o ponerse en contacto con el equipo del centro.
- **Flujo:** El usuario accede a la sección de contacto → visualiza el mapa interactivo, la dirección postal, el teléfono y los horarios de atención → utiliza los datos para contactar o desplazarse al centro.

- **Resultado:** El usuario localiza el centro sin dificultad y conoce todos los canales de contacto disponibles.

CU-03 — Gestionar contenidos desde el CMS

- **Actor:** Usuario interno del centro (dirección, secretaría, docentes)
- **Descripción:** Un miembro del equipo del centro accede al panel de administración de Strapi para crear, editar o eliminar contenidos del sitio web sin necesidad de conocimientos técnicos. Este caso de uso es el más crítico del sistema desde el punto de vista del cliente, ya que la autonomía de gestión era uno de los requisitos no negociables desde el inicio del proyecto.
- **Flujo:** El usuario accede al panel de Strapi con sus credenciales → selecciona el tipo de contenido a modificar → realiza los cambios en los campos del formulario → guarda en modo borrador → revisa el resultado → publica los cambios.
- **Resultado:** Los cambios se reflejan de forma inmediata en el sitio web público sin necesidad de intervención técnica ni de redesplegar la aplicación.

CU-04 — Crear o modificar una página con módulos

- **Actor:** Usuario interno del centro
- **Descripción:** El administrador de contenidos necesita crear una nueva página o modificar la estructura de una existente, añadiendo, reordenando o eliminando módulos de la Dynamic Zone. Este caso de uso refleja la principal ventaja de la arquitectura modular adoptada: cualquier página del sitio puede construirse y reorganizarse visualmente desde el CMS sin tocar el código.
- **Flujo:** El usuario accede a Strapi → selecciona la colección Pages → crea una nueva entrada o edita una existente → añade módulos desde la Dynamic Zone (hero, grid de tarjetas, CTA, mapa, acordeón, etc.) → configura los campos de cada módulo (textos, imágenes, colores, botones) → reordena los módulos arrastrándolos si es necesario → guarda y publica.
- **Resultado:** La nueva página queda accesible en el frontend a través de su ruta configurada, con el contenido y la estructura definidos desde el CMS.

CU-05 — Mantener y actualizar el sistema

- **Actor:** Administrador técnico (desarrollador o responsable técnico del mantenimiento)
- **Descripción:** El desarrollador accede al código fuente del proyecto para actualizar dependencias, corregir errores, añadir nuevos tipos de módulo o realizar cambios en la configuración del sistema. Este caso de uso está pensado para intervenciones técnicas puntuales que van más allá de la gestión de contenidos, y que no debería necesitar el equipo del centro en el uso cotidiano del sitio.
- **Flujo:** El administrador clona el repositorio de GitHub → realiza los cambios necesarios en Next.js o Strapi → prueba el resultado en local → hace commit con un mensaje descriptivo → despliega los cambios en producción.
- **Resultado:** El sistema se mantiene actualizado, seguro y funcional, y cualquier nuevo módulo o mejora queda disponible para el equipo del centro desde el panel de administración.

3.2 Requisitos funcionales

Los requisitos funcionales definen qué debe hacer el sistema. Se organizan por ámbito de funcionalidad y responden directamente a las necesidades identificadas durante el análisis y la toma de requisitos con el centro.

Gestión de páginas y contenidos

- **RF-01:** El sistema debe permitir crear páginas con una ruta personalizada configurable desde el CMS, de forma que el equipo del centro pueda añadir nuevas secciones al sitio sin intervención técnica.
- **RF-02:** El sistema debe renderizar dinámicamente el contenido de cada página a partir de los módulos configurados en la Dynamic Zone de Strapi, de modo que la estructura visual de cada página pueda modificarse desde el panel de administración sin tocar el código.
- **RF-03:** El sistema debe soportar al menos los siguientes tipos de módulo: hero avanzado, grid de tarjetas con imagen, CTA doble, cómo llegar, sección de texto, acordeón, galería, vídeo incrustado, línea de tiempo y pestañas de contenido. Estos módulos cubren la totalidad de los tipos de contenido identificados durante el análisis de la web existente y el estudio comparativo con otros centros.
- **RF-04:** El sistema debe permitir añadir, reordenar y eliminar módulos dentro de una página desde el panel de administración, sin necesidad de modificar el código. El orden visual de los módulos en la página debe corresponder directamente al orden en que aparecen en la Dynamic Zone de Strapi.
- **RF-05:** El sistema debe mostrar los contenidos actualizados en el frontend tras publicar cambios en Strapi, sin necesidad de redesplegar la aplicación. Esto es un requisito crítico para que el equipo del centro pueda gestionar el sitio de forma verdaderamente autónoma.

Navegación y enrutamiento

- **RF-06:** El sistema debe resolver rutas dinámicas de cualquier profundidad —por ejemplo, /escola/historia-del-centre o /etapes/primaria— a partir del campo rutaCompleta configurado en Strapi, permitiendo estructuras de navegación jerárquicas sin limitaciones.
- **RF-07:** El sistema debe implementar un mecanismo de fallback por slug cuando no se encuentre una página por rutaCompleta, para garantizar compatibilidad con contenidos existentes y evitar errores 404 en caso de rutas mal configuradas.
- **RF-08:** El sistema debe mostrar una página de error clara y controlada cuando no se encuentre ninguna página que coincida con la ruta solicitada, sin exponer información técnica al usuario final.

Medios y recursos

- **RF-09:** El sistema debe mostrar correctamente las imágenes asociadas a los módulos, incluyendo las imágenes de tarjetas dentro de componentes anidados en la Dynamic Zone, resolviendo correctamente las relaciones de populate en la API de Strapi.
- **RF-10:** El sistema debe construir las URLs de los recursos multimedia a partir de la URL base de Strapi configurada como variable de entorno, evitando rutas relativas rotas al cambiar de entorno entre local y producción.
- **RF-11:** El sistema debe soportar la incrustación de vídeos externos de plataformas como YouTube o Vimeo mediante una URL configurada desde el CMS, sin necesidad de alojar los vídeos en el propio servidor.

Administración y seguridad

- **RF-12:** El acceso al panel de administración de Strapi debe estar protegido por autenticación con usuario y contraseña, impidiendo el acceso no autorizado a la gestión de contenidos.
- **RF-13:** El sistema debe permitir gestionar roles y permisos en Strapi para diferenciar entre administradores con acceso completo y editores de contenido con permisos limitados, de forma que distintos miembros del equipo del centro puedan acceder al CMS con el nivel de permisos adecuado a su rol.

- **RF-14:** La API de Strapi debe exponer únicamente los endpoints necesarios para el consumo público del frontend, manteniendo el resto de endpoints restringidos y fuera del alcance de usuarios no autenticados.

3.3 Requisitos no funcionales

Los requisitos no funcionales definen cómo debe comportarse el sistema más allá de sus funciones concretas. En este proyecto cobran especial relevancia los relacionados con la accesibilidad y la usabilidad, dado que uno de los problemas principales de la web anterior era precisamente el incumplimiento de estos estándares, y que el cliente final es un equipo sin perfil técnico que necesita poder trabajar con el sistema de forma autónoma y sin fricciones.

Rendimiento

- **RNF-01:** El sitio web debe obtener una puntuación igual o superior a 85 en la categoría de rendimiento de Google Lighthouse en dispositivos móviles, garantizando una experiencia de carga aceptable para todos los usuarios independientemente del dispositivo que utilicen.
- **RNF-02:** El tiempo de carga de la página de inicio no debe superar los 3 segundos en una conexión estándar de banda ancha, un umbral crítico a partir del cual se incrementa significativamente la tasa de abandono del sitio.
- **RNF-03:** Las imágenes deben servirse en formato optimizado y con atributos de dimensiones definidos para evitar cambios de layout durante la carga (CLS), aprovechando el componente Image de Next.js que gestiona automáticamente la optimización y el lazy loading.

Accesibilidad

- **RNF-04:** El sitio debe cumplir con el nivel AA del estándar WCAG 2.1, garantizando contraste de color suficiente, navegación completa por teclado y textos alternativos en todas las imágenes. Este requisito responde directamente a uno de los incumplimientos más graves detectados en la web anterior del centro.

- **RNF-05:** Todos los elementos interactivos deben ser accesibles mediante lector de pantalla, con roles y etiquetas ARIA correctamente implementados donde sea necesario, especialmente en los componentes más complejos como el acordeón, las pestañas de contenido y el menú de navegación.
- **RNF-06:** El sitio debe ser completamente funcional en dispositivos móviles, tabletas y ordenadores de escritorio, con un diseño responsivo adaptado a cada resolución. La falta de adaptación móvil era uno de los problemas más evidentes de la web anterior y uno de los primeros puntos señalados durante el análisis inicial.

Usabilidad

- **RNF-07:** La estructura de navegación debe permitir al usuario encontrar cualquier sección del sitio en un máximo de tres clics desde la página de inicio, garantizando una experiencia de navegación fluida y sin fricciones para las familias, el principal perfil de usuario del sitio.
- **RNF-08:** El panel de administración de Strapi debe permitir a un usuario sin conocimientos técnicos crear o editar una página en menos de 10 minutos tras una formación básica inicial. Este requisito es uno de los más importantes del proyecto dado que la autonomía de gestión del centro era una condición no negociable desde el inicio, y para verificarlo se elaboró el manual de uso del CMS orientado específicamente a usuarios no técnicos.
- **RNF-09:** Los mensajes de error del frontend deben ser claros y comprensibles para el usuario final, sin mostrar información técnica ni trazas de depuración, garantizando una experiencia controlada incluso cuando algo falla.

Seguridad

- **RNF-10:** El sitio debe incluir política de privacidad, aviso legal y gestión de cookies conforme al Reglamento General de Protección de Datos (RGPD), subsanando uno de los incumplimientos legales más graves detectados en la web anterior del centro.
- **RNF-11:** La comunicación entre el frontend y la API de Strapi debe realizarse siempre mediante HTTPS en el entorno de producción, garantizando que los datos transmitidos entre ambas capas no puedan ser interceptados.

- **RNF-12:** Las credenciales de acceso al CMS y las variables de entorno sensibles —URL de la API, tokens de autenticación, configuración de base de datos— deben gestionarse mediante ficheros .env y no deben incluirse en el repositorio de código bajo ninguna circunstancia.

Mantenibilidad

- **RNF-13:** El código del frontend debe estar organizado en componentes reutilizables y documentado de forma que otro desarrollador pueda entender la estructura del proyecto sin asistencia, lo que resulta especialmente relevante dado que el centro podría necesitar incorporar a otro técnico en el futuro para ampliar o mantener el sitio.
- **RNF-14:** Añadir un nuevo tipo de módulo al sistema debe requerir únicamente tres pasos: crear el componente React en Next.js, registrar el content-type correspondiente en Strapi y añadir el caso en el switch de ModuleRenderer, sin necesidad de modificar la lógica de enrutamiento ni de fetching. Esta extensibilidad fue un criterio de diseño arquitectónico desde el inicio del proyecto.
- **RNF-15:** El proyecto debe gestionarse con control de versiones Git, con un historial de commits claro y descriptivo que refleje el progreso del desarrollo y permita revertir cambios en caso de error.

3.4 Análisis de alternativas tecnológicas

Antes de tomar las decisiones técnicas definitivas se evaluaron distintas alternativas para las dos capas principales del sistema: el frontend y el CMS. Este análisis fue especialmente importante dado que una de las condiciones del proyecto era que la solución resultante pudiera ser gestionada de forma autónoma por el equipo del centro, sin conocimientos técnicos, lo que descartaba de entrada cualquier opción que priorizara la complejidad técnica sobre la usabilidad del panel de administración. Cabe mencionar también que en las primeras semanas del proyecto el centro había solicitado inicialmente que la web se desarrollara en WordPress, petición que se evaluó y se descartó por las razones que se detallan a continuación.

Frontend: Next.js vs. alternativas

Se consideraron tres opciones principales para el desarrollo del frontend.

Create React App (CRA) fue descartado desde el principio porque está orientado a aplicaciones de página única (SPA) y no ofrece renderizado en servidor ni generación estática de forma nativa. Esto penaliza directamente el SEO y el rendimiento inicial de carga, dos factores críticos para una web institucional pública cuyo objetivo es ser encontrada por familias que buscan información sobre el centro. Además, CRA lleva años en desuso en el ecosistema React y no recibe actualizaciones activas, lo que lo convierte en una opción sin futuro a largo plazo.

Gatsby es un generador de sitios estáticos basado en React con buen rendimiento y SEO. Sin embargo, su modelo de construcción estática implica que cualquier actualización de contenido requiere reconstruir y redesplegar el sitio completo, lo que lo hace directamente incompatible con el requisito de que los cambios publicados en Strapi se reflejen en el frontend sin necesidad de redespliegue. Esto habría eliminado la autonomía de gestión del centro, uno de los objetivos no negociables del proyecto. Además, su ecosistema de plugins y su curva de configuración son más complejos para el perfil de un proyecto de prácticas.

Next.js con App Router fue la opción elegida por varios motivos decisivos. Soporta renderizado en servidor (SSR) y renderizado estático incremental (ISR), lo que permite

combinar rendimiento óptimo con actualización de contenidos en tiempo real sin necesidad de redespliegue. Su sistema de enrutamiento basado en el sistema de ficheros es intuitivo y escalable, y la ruta dinámica [[...slug]] permite resolver rutas de cualquier profundidad a partir de los datos de Strapi con muy poco código. La comunidad y la documentación son extensas, y es la tecnología más demandada del mercado React en el momento del proyecto, lo que aporta un valor formativo adicional relevante para el perfil DAM.

CMS: Strapi vs. alternativas

Se evaluaron cuatro opciones de gestión de contenidos, teniendo siempre presente que el panel de administración debía ser usable de forma autónoma por el equipo del centro sin formación técnica específica.

WordPress es la plataforma web más utilizada del mundo y fue la primera opción planteada por el propio centro al inicio del proyecto. Sin embargo, tras evaluarla en detalle se descartó por varias razones. Su arquitectura monolítica, basada en PHP con temas y plugins, está pensada para gestionar frontend y backend de forma integrada, lo que la hace menos adecuada para un frontend moderno en React. Usarlo en modo headless mediante la API REST de WordPress es técnicamente posible, pero añade fricción innecesaria y limita las posibilidades de estructuración de contenidos en comparación con un CMS diseñado nativamente para ese propósito. Además, la gestión de un WordPress autohospedado implica una mayor carga de mantenimiento y actualizaciones de seguridad que el centro no estaba en condiciones de asumir.

Contentful es un CMS headless de pago con una interfaz muy cuidada y buena integración con Next.js. Fue descartado porque su plan gratuito tiene limitaciones de contenido y número de usuarios que podrían resultar insuficientes para el centro a medio plazo, y porque introduce una dependencia permanente de un servicio externo de pago para una institución educativa con presupuesto limitado. En el contexto de este proyecto, donde el centro ya había manifestado que no quería asumir costes adicionales, esta dependencia era un riesgo inaceptable.

Sanity es un CMS headless moderno con un modelo de datos muy flexible y una experiencia de desarrollo excelente. Se valoró positivamente desde el punto de vista técnico, pero su

curva de aprendizaje inicial es más pronunciada que la de Strapi, su sistema de consultas GROQ es propietario y requiere aprendizaje específico, y su interfaz de administración es menos intuitiva para usuarios sin perfil técnico que la de Strapi, lo que lo hacía menos adecuado para el requisito de gestión autónoma por parte del centro.

Strapi fue la opción elegida por ser de código abierto, autoalojable sin coste de licencia, y por contar con una interfaz de administración clara e intuitiva especialmente pensada para usuarios no técnicos. El motivo técnico más determinante fue su soporte nativo para Dynamic Zones, que es exactamente el modelo de datos necesario para un sistema de páginas compuestas por módulos reutilizables configurables desde el CMS. Su API REST es estándar, bien documentada y fácil de consumir desde Next.js, y su comunidad es la más activa dentro de los CMS headless de código abierto. La combinación de todos estos factores lo convirtió en la opción que mejor encajaba con los requisitos técnicos, de usabilidad y de viabilidad económica del proyecto.

Servicios de infraestructura en producción: alternativas evaluadas

Más allá del frontend y el CMS, la puesta en producción del proyecto requería decidir tres servicios adicionales de infraestructura: base de datos, almacenamiento de imágenes y monitorización. En todos los casos se evaluaron las alternativas disponibles teniendo en cuenta el requisito de coste cero para el proyecto.

Para la base de datos, Render ofrece su propio servicio PostgreSQL gestionado, pero su plan gratuito expira a los 90 días y elimina los datos almacenados, lo que lo hace inviable para un proyecto en producción. PlanetScale fue descartado por haber eliminado su plan gratuito. Se eligió Supabase por ofrecer PostgreSQL gestionado de forma permanente en su plan gratuito, con conexión mediante session pooler compatible con Strapi y sin limitaciones de tiempo.

Para el almacenamiento de imágenes, la opción por defecto de Strapi es guardarlas en el sistema de ficheros local del servidor, lo que no es viable en Render porque dicho sistema es efímero y se pierde en cada redespliegue. AWS S3 y Cloudflare R2 son alternativas técnicamente válidas pero más complejas de configurar para el perfil de este proyecto. Se eligió Cloudinary por su integración nativa con Strapi mediante proveedor oficial, su plan

gratuito generoso y su capacidad de optimización automática de imágenes en la entrega, lo que aporta además un beneficio directo de rendimiento al sitio sin configuración adicional.

Para la monitorización, se evaluó Freshping como alternativa, pero se eligió UptimeRobot por ser la solución más extendida para el caso de uso concreto de mantener activas instancias en planes gratuitos de Render, con configuración en menos de cinco minutos y alertas por correo electrónico sin coste. El monitor se configuró con un ping cada cinco minutos al endpoint del backend, garantizando que el servicio permanezca disponible en todo momento sin intervención manual.

4. Diseño

4.1 Arquitectura del sistema

La arquitectura del sistema se basa en un modelo cliente-servidor desacoplado, conocido como arquitectura headless, donde el frontend y el backend operan de forma completamente independiente y se comunican exclusivamente a través de una API REST. Este modelo fue elegido precisamente porque permite que el equipo del centro gestione los contenidos desde un panel de administración sin necesidad de tocar el código del frontend, y porque facilita el mantenimiento y la evolución independiente de cada capa del sistema.

El sistema se compone de dos capas principales:

Backend — Strapi CMS

Strapi actúa como gestor de contenidos headless y es la capa responsable de almacenar y exponer todos los datos del sitio. Se despliega en Render como servicio independiente y expone una API REST que el frontend consume para obtener el contenido de cada página.

La base de datos que utiliza Strapi es PostgreSQL, gestionada mediante Supabase, un servicio de base de datos en la nube que proporciona conexión mediante session pooler y elimina la necesidad de administrar infraestructura de base de datos propia. Las imágenes del sitio no se almacenan en el servidor de Strapi sino en Cloudinary, un servicio de gestión y entrega de medios en la nube que optimiza automáticamente el formato y el tamaño de cada imagen en función del dispositivo que la solicita, mejorando el rendimiento y descargando al servidor de Render del coste de servir archivos estáticos. Para mantener activa la instancia de Strapi en Render, que en el plan gratuito entra en suspensión tras un periodo de inactividad, se configuró UptimeRobot con un monitor de tipo ping que consulta el endpoint del backend cada cinco minutos, garantizando que el servicio esté disponible en todo momento sin intervención manual.

La estructura de datos principal se organiza en torno a un content-type llamado Pages, donde cada registro representa una página del sitio web. Cada página contiene una Dynamic

Zone llamada Modulos, que es una lista ordenada de bloques de contenido de distintos tipos —hero, tarjetas, CTA, acordeón, galería, etc.— que el equipo del centro puede añadir, reordenar y eliminar desde el panel de administración sin modificar el código.

Frontend — Next.js con App Router

El frontend está desarrollado con Next.js utilizando el App Router y se despliega en Vercel como servicio independiente. Es la capa que el usuario final ve y con la que interactúa, y su única fuente de datos es la API REST de Strapi.

El sistema de enrutamiento se basa en una ruta dinámica [[...slug]] que captura cualquier URL del sitio y ejecuta la lógica de resolución de páginas: consulta la API de Strapi con la ruta completa solicitada, y si no encuentra coincidencia intenta un segundo intento por slug como mecanismo de fallback. Si ninguna de las dos búsquedas devuelve resultado, se muestra una página de error controlada.

Una vez obtenidos los datos de la página desde Strapi, el componente ModuleRenderer recorre la lista de módulos de la Dynamic Zone y renderiza el componente React correspondiente a cada uno, pasándole sus datos como props. Cada módulo es un componente independiente y autocontenido, lo que facilita el mantenimiento y permite añadir nuevos tipos de módulo al sistema sin modificar la lógica de enrutamiento ni de fetching.

Todas las llamadas a la API de Strapi se realizan desde el lado del servidor mediante Server Components de Next.js, lo que garantiza que los datos estén disponibles en el momento del renderizado inicial, mejora el rendimiento percibido y evita exponer la URL interna de la API al navegador del usuario.

Flujo general de una petición

Cuando un usuario accede a cualquier URL del sitio, el flujo que sigue el sistema es el siguiente: Next.js recibe la petición y la ruta dinámica [[...slug]] captura los segmentos de la URL. Se construye la rutaCompleta y se consulta la API de Strapi filtrando por ese valor. Si se encuentra la página, se devuelven todos sus datos incluyendo los módulos de la Dynamic Zone con todos sus campos y relaciones. ModuleRenderer itera por los módulos y renderiza

cada componente con sus datos. El usuario recibe la página completamente renderizada en el servidor, lista para mostrarse sin necesidad de esperar a cargas adicionales de datos en el cliente.

Segunda entrega — NODES2 (xtecagora)

Como se explicó en la introducción, el proyecto cuenta con una segunda entrega desarrollada sobre la plataforma NODES2 de la Generalitat de Catalunya, que es la versión que el centro utilizará en producción. Esta versión tiene una arquitectura completamente diferente y mucho más sencilla: NODES2 es una plataforma gestionada basada en WordPress que no requiere configuración de servidor, despliegue ni mantenimiento técnico por parte del centro. El diseño de esta versión es más básico y está adaptado a las posibilidades y limitaciones de la plataforma, pero mantiene la estructura de navegación y la identidad visual definidas durante las fases de análisis y diseño del proyecto.

4.2 Modelo de datos

El modelo de datos del sistema está definido íntegramente en Strapi y se organiza en torno a dos content-types principales: Pages, de tipo Collection, y Site-setting, de tipo Single Type. Toda la estructura de contenido del sitio web —páginas, secciones, textos, imágenes y configuración global— se gestiona a través de estos dos elementos.

Pages (Collection Type)

Es el content-type central del sistema. Cada registro representa una página del sitio web y contiene los siguientes campos:

- **title:** Texto corto. Título identificativo de la página, usado en el panel de administración.
- **slug:** Texto corto con valor único. Identificador de la página usado como mecanismo de fallback en el enrutamiento cuando no se encuentra la página por rutaCompleta.
- **rutaCompleta:** Texto corto. Ruta completa de la página tal como aparece en la URL del sitio, por ejemplo /escola/historia-del-centre. Es el campo principal que usa el frontend para resolver qué página mostrar en cada ruta.
- **Modulos:** Dynamic Zone. Es el campo más importante del content-type. Contiene la lista ordenada de módulos de contenido que componen la página. Cada módulo es de un tipo distinto y tiene sus propios campos. El orden de los módulos en esta zona determina directamente el orden visual en que aparecen en la página.

Site-setting (Single Type)

Contiene la configuración global del sitio, campos que se aplican a todas las páginas y no pertenecen a ninguna en concreto. Incluye el nombre del centro, los datos de contacto, los enlaces a redes sociales y otros parámetros generales como el texto del footer o el logotipo del sitio. Al ser un Single Type solo existe un único registro de este tipo, accesible desde cualquier parte del frontend.

Módulos de la Dynamic Zone

Cada módulo es un componente de Strapi con sus propios campos. Todos los módulos que contienen imágenes las gestionan mediante un campo de tipo Media que apunta al gestor de archivos integrado de Strapi. A continuación se describen todos los módulos implementados:

HeroSection — Cabecera estándar con imagen de fondo. Campos: título, subtítulo, descripción, imagen de fondo (Media), mostrarBreadcrumbs (booleano), texto del botón y URL del botón.

HeroAvanzado — Cabecera ampliada pensada para páginas principales con más opciones de personalización. Campos: imagen de fondo (Media), dos líneas de título, descripción, badge con texto e icono, y una lista de botones destacados con texto y URL cada uno.

SecciónTexto — Bloque de texto de contenido general. Campos: título opcional, contenido en formato Rich Text, alineación (izquierda, centro o derecha) y color de fondo.

CajaDestacada — Bloque con color de fondo, icono y texto para resaltar información importante. Campos: icono en formato emoji, título, descripción en Rich Text, disposición (icono a la izquierda o icono arriba), estilo de fondo (sólido o gradiente), tema de color y campo de color HEX opcional para personalizaciones avanzadas.

TarjetasGrid — Cuadrícula de tarjetas de contenido. Campos del módulo: título de la sección y número de columnas (1, 2, 3 o 4). Contiene una lista de componentes anidados de tipo CardItem, cada uno con: icono en formato emoji, título, descripción y enlace opcional.

CTASection — Bloque de llamada a la acción. Campos: título, descripción, texto y URL del primer botón, texto y URL del segundo botón.

Acordeón — Sección de preguntas y respuestas plegables. Campos: título de la sección y una lista de elementos anidados, cada uno con una pregunta y una respuesta en Rich Text.

PestañasContenido — Contenido organizado en pestañas navegables. Campos: título de la sección y una lista de pestañas anidadas, cada una con título, icono y contenido en Rich Text.

Galería — Cuadrícula de imágenes. Campos: título de la sección y una lista de imágenes en formato Media.

MapaUbicación — Sección con mapa interactivo y datos de ubicación del centro. Campos: título, dirección, coordenadas y texto descriptivo.

VideoIncrustado — Sección para incrustar vídeos externos. Campos: título, URL del vídeo (YouTube o Vimeo) y texto descriptivo opcional.

LíneaTiempo — Visualización de pasos o etapas en formato cronológico. Campos: título de la sección y una lista de elementos anidados, cada uno con fecha o etiqueta, título y descripción.

CuadrículaEstadísticas — Sección para mostrar datos o cifras destacadas. Campos: título de la sección y una lista de estadísticas anidadas, cada una con valor numérico, etiqueta descriptiva e icono opcional.

Relaciones y gestión de medios

El sistema no utiliza relaciones entre content-types distintos más allá de las que Strapi gestiona internamente. Las imágenes se gestionan a través del Media Library integrado de Strapi, donde se almacenan todos los archivos subidos al sistema. Cada campo de tipo Media en cualquier módulo apunta a un archivo de esta biblioteca, y la URL pública de cada imagen se construye en el frontend concatenando la URL base de Strapi con la ruta relativa devuelta por la API, lo que garantiza que las imágenes funcionen correctamente tanto en local como en producción.

4.3 Diseño de interfaz

El diseño de interfaz del proyecto se desarrolló en dos etapas diferenciadas: una fase de prototipado en Figma previa al desarrollo, y la implementación real de los componentes visuales durante la fase de desarrollo. Ambas etapas se guiaron en todo momento por la guía de estilo institucional elaborada al inicio del proyecto, que estableció las bases visuales y de identidad que debía seguir todo el sitio.

Guía de estilo institucional

Antes de diseñar ninguna pantalla se definió la guía de estilo institucional del centro, un documento de referencia que unifica todos los elementos visuales del sitio y garantiza la coherencia gráfica en cualquier página o componente. La guía surgió de la necesidad detectada durante el análisis: la web anterior del centro no tenía ningún criterio visual sistematizado, lo que generaba incoherencias tipográficas y cromáticas a lo largo de todo el sitio.

Logotipo: El logotipo original del centro, basado en un degradado dorado con detalles artesanales, presentaba limitaciones importantes en entornos digitales: perdía nitidez en tamaños reducidos, no funcionaba bien en versiones monocromas y era difícil de integrar en una interfaz moderna. Se propuso una evolución hacia un diseño vectorial limpio en formato SVG que mantiene el símbolo central de la rama y el círculo pero lo adapta a un lenguaje gráfico contemporáneo, escalable a cualquier tamaño y compatible con cualquier fondo. Sin embargo, tras presentar la propuesta a la dirección del centro, la directora indicó que no le convencía el nuevo diseño y prefería mantener el logotipo original, por lo que la versión final del sitio utiliza el logotipo existente del centro.

Paleta de colores: Se definieron tres propuestas basadas en tonos verde oliva, conectando con el símbolo del olivar que representa al centro. La paleta elegida utiliza un verde oliva equilibrado como color institucional principal, complementado con tonos neutros para fondos y textos, y variantes más oscuras y claras para elementos interactivos y cabeceras. Todos los pares de color cumplen con los requisitos de contraste del estándar WCAG 2.1 nivel AA.

Tipografía: Se establecieron dos familias obtenidas de Google Fonts: Montserrat para títulos y elementos de cabecera, aportando presencia visual y personalidad, y Open Sans para textos de cuerpo y elementos de interfaz, priorizando la legibilidad en pantalla.

Iconografía: Se adoptó un sistema basado en emojis para los módulos gestionados desde Strapi, lo que permite al equipo del centro añadir y cambiar iconos desde el panel de administración sin necesidad de gestionar archivos ni conocer ningún sistema técnico. Para los elementos de interfaz fijos del frontend se utilizan iconos vectoriales SVG integrados directamente en los componentes.

Criterios fotográficos: Todas las imágenes del sitio provienen del propio centro educativo, garantizando autenticidad y cercanía, siguiendo criterios de selección basados en luminosidad, composición y representatividad de la vida escolar del centro.

Prototipo Figma

Con la guía de estilo definida, se creó un prototipo completo de la web en Figma antes de iniciar el desarrollo. Este prototipo sirvió como referencia visual para validar el diseño con el centro antes de escribir una sola línea de código, evitando tener que rehacer trabajo de desarrollo por cambios de criterio estético. El prototipo definió la estructura visual de las páginas principales del sitio: página de inicio, páginas de etapas educativas, página de contacto y páginas de contenido genérico, estableciendo para cada una la distribución de los bloques de contenido, la jerarquía visual entre los elementos, el comportamiento del menú de navegación y la adaptación del diseño a distintas resoluciones de pantalla.

El proceso de validación del prototipo con el centro fue uno de los primeros momentos en que se hizo evidente la dificultad de obtener feedback fluido por parte de la dirección. A pesar de que el prototipo estuvo listo en los plazos previstos, la directora tardó varias semanas en revisarlo y sus comentarios se limitaron a aspectos puntuales como el logotipo, sin entrar en valoraciones sobre la estructura o la navegación, lo que obligó a avanzar hacia el desarrollo asumiendo que el diseño general era aceptado.

Componentes de interfaz

A partir de la guía de estilo y el prototipo, se implementaron los componentes de interfaz que forman el sistema visual del sitio, todos diseñados con un enfoque mobile-first y adaptados a cualquier resolución de pantalla desde móvil hasta escritorio.

Layout global: El sitio cuenta con una barra de navegación superior con el logotipo del centro, el menú principal con submenús desplegables y un menú hamburguesa para dispositivos móviles. El pie de página incluye los datos de contacto del centro, los enlaces a redes sociales, los vínculos institucionales con la Generalitat de Catalunya y los enlaces a política de privacidad y aviso legal.

Sistema de botones: Se definieron dos variantes principales: el botón primario con fondo en el color institucional verde oliva y el botón secundario con borde y fondo transparente. Ambos tienen estados de hover y focus claramente diferenciados y cumplen con los requisitos de contraste de WCAG 2.1.

Tarjetas de contenido: Las tarjetas del módulo TarjetasGrid siguen un diseño consistente con icono en la parte superior, título en Montserrat y descripción en Open Sans, con el número de columnas configurable desde Strapi para adaptarse al tipo de contenido de cada sección.

Módulos hero: Los módulos de cabecera ocupan el ancho completo de la pantalla con una imagen de fondo, una capa de oscurecimiento para garantizar el contraste del texto superpuesto, y el título, subtítulo y botones centrados verticalmente. El HeroAvanzado añade un badge informativo y mayor flexibilidad en la disposición de los elementos.

Breadcrumbs: Las páginas interiores muestran una ruta de navegación en migas de pan bajo el hero, que indica al usuario en qué sección del sitio se encuentra y le permite navegar hacia niveles superiores con un solo clic.

Formulario de contacto: La página de contacto incluye un formulario con campos de nombre, correo electrónico, asunto y mensaje, con validación en el frontend y mensajes de error claros y comprensibles para el usuario.

Página de error: Se implementó una página 404 personalizada que mantiene la identidad visual del sitio, informa al usuario de que la página no existe y le ofrece un enlace para volver a la página de inicio.

5. Desarrollo

5.1 Estructura del proyecto

El proyecto está organizado en un único repositorio de GitHub que contiene las dos capas del sistema en carpetas separadas: el frontend desarrollado con Next.js y el backend con Strapi. Esta estructura de monorepo facilita la gestión del proyecto al centralizar todo el código en un único repositorio con control de versiones compartido, manteniendo al mismo tiempo una separación clara entre las dos capas del sistema. El repositorio contiene también dos carpetas adicionales, pshad/ y pshadcn/, que corresponden a proyectos complementarios de exploración de componentes de interfaz con shadcn/ui utilizados como referencia durante el desarrollo.

Carpeta backend/

Contiene la instalación completa de Strapi 5.31.2 con toda la configuración del CMS. Incluye la definición de los content-types Pages y Site-setting, los componentes reutilizables de la Dynamic Zone con sus esquemas de campos, la configuración del servidor, la gestión de permisos y roles, y la base de datos PostgreSQL con todos los contenidos reales del centro cargados durante la fase de desarrollo. El backend está desarrollado sobre Node.js 18 y desplegado en Render en producción.

Carpeta frontend/

Contiene el proyecto Next.js 16 con React 19, TypeScript y App Router. Dentro se organiza en las siguientes subcarpetas:

app/ — Directorio principal de rutas de Next.js. Contiene la ruta dinámica **[...slug]/page.tsx** que captura cualquier URL del sitio y ejecuta la lógica de resolución de páginas contra la API de Strapi, el archivo **page.tsx** de la página de inicio, el **layout.tsx** global que carga el header, la navegación y el footer en todas las páginas, los estilos globales en **globals.css** y el favicon. Contiene también una serie de carpetas legacy —admissio/, comunitat/, contacte/, etapes/, legal/, lescola/, serveis/ y vida-escolar/— que corresponden a la arquitectura anterior del

proyecto, cuando cada sección tenía su propia carpeta estática dentro de app/. Estas carpetas quedaron obsoletas cuando se migró completamente al sistema dinámico de Strapi con el segmento **[...slug]** y pueden ser eliminadas sin afectar al funcionamiento del sistema.

components/ — Contiene todos los componentes React del proyecto, organizados en tres grupos:

- **Componentes de layout global:** Header.tsx, Footer.tsx, Nav.tsx y TopBar.tsx, que se cargan en todas las páginas a través del layout global y proporcionan la estructura común del sitio.
- **modules/** — Un componente por cada tipo de módulo de la Dynamic Zone, más el componente ModuleRenderer.tsx que actúa como distribuidor central. Los módulos implementados son: HeroSection, HeroAvanzado, TarjetasGrid, CtaSection, SeccionTexto, CajaDestacada, ImagenTexto, LineaTiempo, Acordeon, CuadriculaEstadisticas, Galeria, VideoIncrustado, PestanasContenido, MapaUbicacion, SeccionBienvenida y ListaObjetivos, un total de dieciséis módulos reutilizables configurables desde Strapi.
- **ui/** — Componentes de interfaz reutilizables que no corresponden a ningún módulo concreto de la Dynamic Zone.

lib/ — Contiene el archivo strapi.ts con todas las funciones de fetching de datos desde la API de Strapi. Aquí se centralizan las llamadas a la API, incluyendo la lógica de resolución de páginas por rutaCompleta y el fallback por slug, la obtención de la configuración global del sitio desde Site-setting y la construcción de las URLs de los recursos multimedia. La URL base de Strapi se gestiona como variable de entorno desde este archivo.

public/ — Recursos estáticos del proyecto: logotipo del centro, favicon, imágenes que no se gestionan desde Strapi y cualquier otro archivo estático servido directamente por Next.js.

styles/ — Contiene los archivos CSS del proyecto organizados por ámbito: globals.css para los estilos globales, y archivos específicos por componente o sección: menu.css, hero.css, highlighted-cards.css, call-to-action.css, map-location.css, footer.css, legal-pages.css y content-pages.css. Esta organización permite localizar y modificar los estilos de cada parte del sitio de forma rápida y sin interferir con el resto del sistema.

5.2 Implementación de funcionalidades

Enrutamiento dinámico

El núcleo del sistema de enrutamiento se basa en un único archivo **[...slug]/page.tsx** dentro del directorio **app/** de Next.js. Este segmento de ruta dinámica captura cualquier URL del sitio, independientemente de su profundidad, y ejecuta la lógica de resolución de páginas contra la API de Strapi. El proceso sigue los siguientes pasos: se construye la rutaCompleta concatenando los segmentos capturados por el slug, se consulta la API de Strapi filtrando por ese valor, y si no se encuentra ninguna coincidencia se realiza un segundo intento filtrando únicamente por el último segmento de la ruta como mecanismo de fallback por slug. Si ninguna de las dos búsquedas devuelve resultado, Next.js renderiza la página de error 404 personalizada. Este sistema permite que cualquier página creada en Strapi con una rutaCompleta configurada sea accesible automáticamente en el frontend sin necesidad de modificar el código de enrutamiento. Cabe destacar que el proyecto conserva también una serie de carpetas legacy dentro de app/ que corresponden a la arquitectura anterior, cuando cada sección tenía su propia ruta estática. Estas carpetas quedaron obsoletas con la migración al sistema dinámico y no intervienen en el funcionamiento actual del sitio.

Fetching de datos desde Strapi

Todas las llamadas a la API de Strapi se realizan mediante fetch nativo de Next.js desde Server Components, lo que garantiza que los datos estén disponibles en el momento del renderizado inicial en el servidor y no sean visibles en el cliente. Las funciones de fetching están centralizadas en el archivo lib/strapi.ts, separando la lógica de obtención de datos de los componentes de presentación. El parámetro más crítico de estas llamadas es populate, que indica a Strapi qué relaciones y componentes anidados debe incluir en la respuesta. Dado que la Dynamic Zone contiene módulos con componentes anidados —como las tarjetas dentro de TarjetasGrid o los elementos dentro del Acordeón— el populate debe ser lo suficientemente profundo para que el frontend reciba todos los datos necesarios en una sola llamada. La resolución de los errores de populate fue una de las partes más complejas del desarrollo, ya que una configuración incorrecta hacía que los datos de los módulos anidados llegaran vacíos al frontend sin ningún mensaje de error explícito. La URL base de

Strapi se gestiona como variable de entorno en un fichero .env.local que no se incluye en el repositorio, permitiendo cambiar entre el entorno local y el de producción sin modificar el código.

ModuleRenderer

El componente ModuleRenderer es el nexo central entre los datos de Strapi y los componentes visuales de React. Recibe como prop la lista de módulos de la Dynamic Zone de una página e itera sobre ella con un map. Para cada módulo, lee el campo **__component** que Strapi incluye automáticamente para identificar su tipo y lo evalúa mediante un switch que selecciona el componente React correspondiente y le pasa todos los datos del módulo como props.
Cada componente de módulo es completamente independiente y autocontenido: recibe sus datos, los renderiza y no tiene ninguna dependencia con el resto de módulos de la página. Esta arquitectura hace que añadir un nuevo tipo de módulo al sistema sea un proceso de tres pasos: crear el componente React en la carpeta components/modules/, registrar el content-type correspondiente en Strapi y añadir el caso en el switch de ModuleRenderer, sin tocar ninguna otra parte del código.

Módulos de la Dynamic Zone

Se han implementado dieciséis módulos en total, cada uno siguiendo el mismo patrón: el componente recibe sus datos como props tipadas con TypeScript, los destructura y los renderiza con Tailwind CSS y estilos específicos desde la carpeta styles/. Los módulos que contienen imágenes construyen la URL completa concatenando la URL base de Strapi con la ruta relativa devuelta por la API. Los módulos con listas de elementos anidados —TarjetasGrid, Acordeon, PestanasContenido y LineaTiempo— iteran sobre sus elementos con un map y renderizan cada uno de forma independiente. Los módulos más destacados por su complejidad de implementación son HeroAvanzado, por la variedad de opciones de personalización que ofrece, MapaUbicacion, que integra un mapa interactivo con las coordenadas del centro configuradas desde Strapi, ImagenTexto, que gestiona distintas disposiciones de imagen y texto configurables desde el CMS, y ListaObjetivos y

SeccionBienvenida, que se desarrollaron para cubrir necesidades de contenido específicas del centro detectadas durante la carga de contenidos reales en la fase 4.

Gestión de imágenes

Las imágenes del sitio se gestionan a través del Media Library de Strapi y se sirven directamente desde el servidor de Strapi en producción. En el frontend se utiliza el componente Image de Next.js para todas las imágenes, lo que proporciona optimización automática del formato y tamaño, lazy loading nativo y la definición de dimensiones que evita los cambios de layout durante la carga. La URL de cada imagen se construye en lib/strapi.ts antes de pasarla a los componentes, centralizando en un único punto la lógica de construcción de URLs multimedia.

Formulario de contacto

El formulario de contacto está conectado directamente a Strapi, que actúa como receptor de los mensajes enviados desde el frontend. El formulario incluye campos de nombre, correo electrónico, asunto y mensaje, con validación en el frontend mediante TypeScript antes de realizar el envío. Los mensajes recibidos quedan almacenados en Strapi y son accesibles desde el panel de administración, permitiendo al equipo del centro consultarlos sin necesidad de acceder a ningún servicio externo.

Configuración global y Site-setting

La configuración global del sitio —nombre del centro, datos de contacto, enlaces a redes sociales y logotipo— se obtiene desde el Single Type Site-setting de Strapi en el layout global de Next.js, de forma que está disponible en todas las páginas del sitio sin necesidad de hacer una llamada adicional en cada una. Esto garantiza que cualquier cambio en los datos de contacto o en los enlaces del footer se refleje de forma inmediata en todo el sitio con una única actualización desde el panel de administración.

5.3 Pruebas

Las pruebas realizadas a lo largo del proyecto han sido de tipo manual y funcional, sin uso de frameworks de testing automatizado. Esta decisión responde a la naturaleza y el alcance del proyecto: tratándose de un proyecto individual de prácticas con un plazo de cinco meses y con una fase de desarrollo intensiva, se priorizó destinar el tiempo disponible al desarrollo de funcionalidades y a la calidad del producto final sobre la implementación de una suite de tests automatizados. Las pruebas se realizaron de forma continua a lo largo de todo el desarrollo, verificando cada módulo y funcionalidad en el momento de su implementación antes de avanzar al siguiente.

Pruebas funcionales

Para cada módulo implementado se verificó de forma manual que los datos configurados en Strapi se renderizaban correctamente en el frontend. Esto implicaba crear un registro de prueba en el panel de administración de Strapi con distintos valores en cada campo, publicarlo y comprobar en el navegador que el componente React mostraba la información esperada con el diseño correcto. Se prestó especial atención a los casos límite: módulos con campos opcionales vacíos, tarjetas sin imagen, secciones sin descripción o botones sin URL, verificando que el componente gestionaba correctamente la ausencia de datos sin romper el layout ni generar errores en consola.

Las pruebas funcionales también cubrieron el sistema de enrutamiento dinámico, comprobando que rutas de distinta profundidad —como /escola, /escola/historia o /etapes/primaria/projectes— se resolvían correctamente contra la API de Strapi y cargaban la página esperada. Se verificó también el comportamiento del mecanismo de fallback por slug, creando escenarios en los que la rutaCompleta no coincidía con ninguna página registrada para confirmar que el sistema intentaba la búsqueda por slug antes de mostrar el error 404.

Pruebas de responsividad

Se comprobó el comportamiento del sitio en distintos tamaños de pantalla utilizando las herramientas de desarrollo del navegador Chrome. Las pruebas cubrieron los breakpoints

principales: móvil (375px), tableta (768px), escritorio medio (1280px) y escritorio grande (1440px). Para cada módulo se verificó que la distribución de los elementos se adaptaba correctamente a cada resolución, que el menú de navegación cambiaba correctamente entre la versión de escritorio con submenús desplegables y la versión móvil con menú hamburguesa, y que las imágenes de los módulos hero no perdían su encuadre ni su capa de oscurecimiento en ninguna resolución.

Pruebas de accesibilidad

Se realizaron pruebas de accesibilidad utilizando la herramienta Lighthouse integrada en las herramientas de desarrollo de Chrome, obteniendo las puntuaciones de accesibilidad de las páginas principales del sitio. Se verificó manualmente la presencia de textos alternativos en todas las imágenes, el contraste suficiente entre el texto y el fondo en todos los módulos, y la navegabilidad completa del sitio mediante teclado, comprobando que todos los elementos interactivos —botones, enlaces, menús desplegables, acordeones y pestañas— eran accesibles sin necesidad de ratón. Se revisaron también los roles y atributos ARIA de los componentes más complejos como el menú de navegación y el acordeón.

Pruebas de rendimiento

Se utilizó Google Lighthouse para medir el rendimiento del sitio en dispositivos móviles y de escritorio, analizando las métricas principales: Largest Contentful Paint (LCP), Cumulative Layout Shift (CLS), First Input Delay (FID) y puntuación global de rendimiento. Las imágenes fueron el principal factor de impacto en las métricas de rendimiento, y su optimización mediante el componente Image de Next.js, que gestiona automáticamente el formato, el tamaño y el lazy loading, fue determinante para mejorar las puntuaciones obtenidas. Se verificó también que ninguna imagen causaba cambios de layout durante la carga al tener definidos sus atributos de dimensiones.

Pruebas de la API de Strapi

Durante el desarrollo se realizaron pruebas directas sobre los endpoints de la API de Strapi para verificar que las respuestas incluían todos los datos necesarios con la configuración de populate correcta. Estas pruebas fueron especialmente importantes en los módulos con

componentes anidados, donde un populate incompleto hacía que los datos llegaran vacíos al frontend sin ningún mensaje de error explícito. Se comprobó que la API devolvía correctamente las imágenes con sus URLs, los componentes anidados de TarjetasGrid, Acordeón y PestañasContenido, y la configuración global de Site-setting.

Pruebas de gestión de contenidos

Una vez completado el desarrollo, se realizó una prueba completa del flujo de gestión de contenidos desde el punto de vista de un usuario no técnico, siguiendo el manual elaborado para el centro. Se verificó que era posible crear una nueva página, añadir y reordenar módulos desde la Dynamic Zone, publicar los cambios y comprobar que el resultado era visible de forma inmediata en el frontend sin necesidad de redesplegar la aplicación. Esta prueba fue especialmente relevante dado que la autonomía de gestión del centro era uno de los requisitos fundamentales del proyecto.

6. Conclusiones

6.1 Conclusiones generales

El proyecto de rediseño de la web de la Escola Lluís Millet ha llegado a su fin con un resultado que va más allá de la entrega técnica en sí. A lo largo de cinco meses de prácticas se ha recorrido el ciclo completo de un proyecto web real: desde la primera reunión con un cliente sin perfil técnico que no tenía claro qué quería, pasando por todas las fases de análisis, diseño y desarrollo, hasta llegar a una entrega final condicionada por las limitaciones y decisiones reales del cliente. Este recorrido es, en sí mismo, uno de los aprendizajes más valiosos del proyecto.

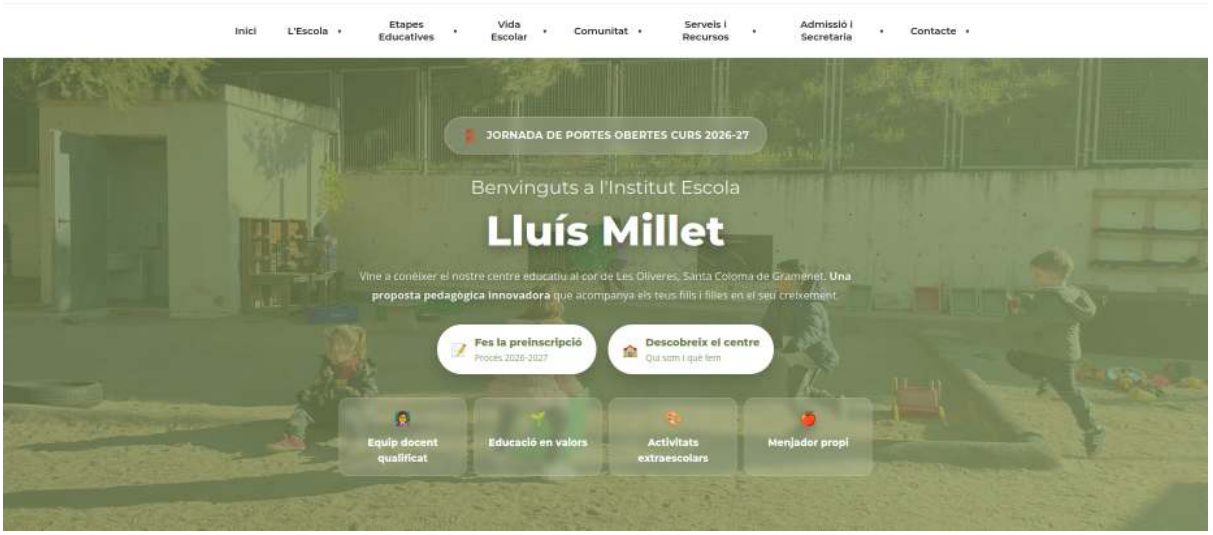
Desde el punto de vista técnico, el proyecto ha demostrado que la combinación de Next.js con App Router y Strapi como CMS headless es una solución sólida, escalable y adecuada para un sitio web institucional de estas características. La arquitectura modular basada en una Dynamic Zone de trece módulos reutilizables permite construir y reorganizar cualquier página del sitio desde el panel de administración sin tocar el código, cumpliendo el requisito fundamental de que el equipo del centro pudiera gestionar los contenidos de forma autónoma. El sistema de enrutamiento dinámico resuelve rutas de cualquier profundidad a partir de los datos de Strapi, y la estructura de componentes del frontend está organizada de forma que añadir nuevos módulos en el futuro requiere un esfuerzo mínimo y no implica modificar la lógica existente.

Desde el punto de vista del proceso, el proyecto ha sido un reflejo fiel de lo que supone trabajar con un cliente real no técnico. La toma de requisitos fue un proceso largo y con más dificultades de las esperadas: los requisitos iniciales cambiaron varias veces, obtener validaciones y feedback de la dirección del centro fue complicado en casi todas las fases, y algunas decisiones importantes —como el diseño del logotipo o la aprobación del prototipo— tuvieron que tomarse sin una confirmación explícita por parte del cliente. La decisión final de no contratar hosting ni dominio propio, y la consiguiente necesidad de desarrollar una segunda versión del sitio sobre NODES2, obligó a replantear la solución de

despliegue en la recta final del proyecto y puso a prueba la capacidad de adaptación a circunstancias imprevistas.

El proyecto presenta dos entregas diferenciadas que juntas representan el trabajo completo realizado durante las prácticas. La primera es la solución técnica completa desarrollada con Next.js y Strapi, desplegada en Vercel y Render, que demuestra la capacidad de diseñar e implementar una arquitectura web moderna y profesional desde cero. La segunda es la adaptación funcional sobre NODES2, una versión más básica que responde a las limitaciones reales del cliente y que es la que el centro utilizará en producción. Lejos de ser un problema, esta dualidad refleja una competencia profesional importante: saber construir la mejor solución técnica posible y saber también adaptarla a las condiciones reales del entorno en el que debe funcionar.

En conjunto, el proyecto ha permitido aplicar de forma práctica y en un contexto real la mayoría de competencias trabajadas a lo largo del ciclo formativo: análisis de requisitos, diseño de interfaces, desarrollo frontend con tecnologías modernas, integración con un CMS headless, despliegue en entornos de producción y documentación técnica orientada a usuarios no técnicos.



Captura de la web de la Escola Lluís Millet desarrollada con Next.js y Strapi, desplegada en Vercel durante la fase de desarrollo.

6.2 Consecución de objetivos

A continuación se analiza el grado de cumplimiento de cada uno de los objetivos específicos definidos al inicio del proyecto, organizados por ámbito.

Análisis y planificación

El objetivo de analizar en profundidad la web actual del centro y compararlo con cuatro centros de referencia se cumplió de forma completa. El análisis de la web existente en blocs.xtec.cat quedó recogido en el documento "Anàlisi Lluís Millet", que identificó los problemas de diseño, accesibilidad, estructura y contenidos del sitio anterior. El estudio comparativo con FEDAC Santa Coloma, Escola Manent, CEIP Serra de Marina y St Peter's School Barcelona permitió extraer buenas prácticas y definir un modelo de referencia adecuado al contexto del centro. La propuesta de nueva arquitectura de contenidos se materializó en el documento "Proposta Organigrama Lluís Millet", que presentó tres alternativas de estructura de navegación para que el centro pudiera elegir la que mejor se ajustara a sus necesidades. El trabajo de toma de requisitos con un cliente no técnico fue más complejo de lo esperado, pero se resolvió mediante reuniones periódicas, formularios de feedback y propuestas iterativas hasta llegar a una arquitectura validada.

Diseño e identidad visual

La guía de estilo institucional se elaboró de forma completa, definiendo logotipo, paleta de colores en tonos verde oliva, tipografías Montserrat y Open Sans, criterios de iconografía y fotografía, y los componentes principales de interfaz. Este objetivo se cumplió con un matiz: la propuesta de evolución del logotipo hacia un diseño vectorial moderno fue rechazada por la directora del centro, que prefirió mantener el logotipo original. Esta decisión, aunque supuso un ajuste respecto al planteamiento inicial, forma parte de la realidad del trabajo con un cliente real y se gestionó sin impacto en el resto del proyecto. El prototipo en Figma se completó y sirvió como referencia visual para el desarrollo, aunque su validación por parte del centro fue más lenta y superficial de lo esperado, lo que obligó a asumir la aprobación implícita del diseño general y avanzar hacia el desarrollo. La interfaz final es responsiva, limpia y accesible, adaptada a cualquier dispositivo y coherente con los valores del centro.

Desarrollo técnico

El frontend con Next.js y App Router se implementó de forma completa, con una arquitectura de componentes modulares escalable y un sistema de enrutamiento dinámico que resuelve rutas de cualquier profundidad. La integración con Strapi como CMS headless se realizó mediante una Dynamic Zone de trece módulos reutilizables, cumpliendo y superando el requisito mínimo de diez tipos de módulo establecido al inicio del proyecto. El cumplimiento de los estándares de accesibilidad WCAG 2.1 y de la normativa RGPD se verificó mediante pruebas con Lighthouse y revisión manual de los elementos interactivos, textos alternativos y contraste de color. El manual de uso del CMS orientado a usuarios no técnicos se elaboró y entregó al centro como parte de la documentación del proyecto.

Resultado

El objetivo de entregar un sitio web funcional, responsivo y mantenible que mejore de forma significativa la comunicación con familias, alumnado y profesorado se cumplió en dos versiones diferenciadas. La versión técnica completa desarrollada con Next.js y Strapi representa la solución óptima desde el punto de vista profesional y demuestra la capacidad de diseñar e implementar una arquitectura web moderna desde cero. La versión sobre NODES2, más básica pero adaptada a las limitaciones reales del centro, es la que el centro utilizará en producción y garantiza que el trabajo del proyecto tiene un impacto real y duradero en la presencia digital del Lluís Millet.

El único objetivo que no se cumplió en los términos originalmente planteados fue el despliegue en producción de la versión Next.js/Strapi como web definitiva del centro, debido a la decisión del cliente de no asumir el coste de hosting y dominio. Sin embargo, esta circunstancia no invalida el trabajo realizado sino que añade una dimensión de aprendizaje adicional al proyecto: la capacidad de adaptar una solución técnica a las limitaciones reales del entorno en el que debe funcionar es una competencia fundamental en el ejercicio profesional del desarrollo de software.

6.3 Valoración de la metodología y planificación

La metodología iterativa e incremental adoptada para el proyecto ha funcionado de forma satisfactoria en términos generales, aunque su aplicación práctica ha revelado tanto sus puntos fuertes como sus limitaciones en el contexto concreto de trabajar con un cliente no técnico y con disponibilidad limitada.

El enfoque de validar cada fase antes de avanzar a la siguiente fue acertado en teoría, pero en la práctica chocó con una de las dificultades más recurrentes del proyecto: la dificultad para obtener feedback del centro en los plazos previstos. En varias ocasiones los entregables —el análisis comparativo, el organigrama de navegación, el prototipo de Figma— estuvieron listos dentro de la planificación prevista pero no pudieron validarse con la dirección durante semanas, bien porque la directora no acudía a las reuniones acordadas, bien porque revisaba los documentos de forma superficial sin dar una respuesta clara. Esta situación generó incertidumbre en momentos clave del proyecto y obligó en más de una ocasión a tomar decisiones de diseño y contenido sin una aprobación explícita por parte del cliente, asumiendo que la ausencia de objeciones equivalía a una validación implícita.

La planificación por fases fue útil para mantener el orden y la dirección del trabajo, pero resultó más rígida de lo deseable en un proyecto con tanta dependencia del cliente. En la práctica, las fases no siempre fueron secuenciales: hubo momentos en que se avanzó en el desarrollo mientras aún se esperaba validación de la fase de diseño, y momentos en que fue necesario retomar decisiones de análisis ya cerradas porque el centro había cambiado de criterio. La lección aprendida es que en proyectos con clientes no técnicos conviene incorporar desde el inicio márgenes de tiempo explícitos para la gestión de validaciones y para absorber los cambios de criterio sin que afecten al conjunto de la planificación.

La gestión del código con Git y GitHub funcionó correctamente a lo largo de todo el proyecto. El trabajo con ramas separadas para cada módulo o funcionalidad nueva permitió mantener el código de producción siempre en un estado estable, y el historial de commits refleja de forma clara el progreso del desarrollo semana a semana. Esta práctica, aunque añade una pequeña carga de trabajo adicional, demostró su valor en varias ocasiones en las que fue necesario revertir cambios o recuperar versiones anteriores de un componente.

La priorización de funcionalidades basada en el valor para el usuario final fue una de las decisiones metodológicas más acertadas del proyecto. Gracias a ella, cuando llegó el momento de replantear la solución de despliegue por la decisión del centro de no contratar hosting, el núcleo del sistema estaba completamente terminado y documentado, lo que permitió abordar la adaptación a NODES2 sin comprometer la calidad del trabajo ya realizado. Si se hubiera seguido un enfoque diferente y se hubieran dedicado recursos a mejoras opcionales antes de tener el núcleo estable, el impacto del cambio de criterio final habría sido mucho mayor.

En cuanto al tiempo, la duración de cinco meses fue suficiente para completar todas las fases del proyecto, aunque el margen final fue más ajustado de lo deseable debido a los retrasos acumulados en las validaciones con el centro y a la necesidad de desarrollar la segunda versión del sitio sobre NODES2 en la recta final. Una planificación futura de un proyecto similar debería contemplar desde el inicio un colchón de tiempo específico para gestionar este tipo de imprevistos, especialmente cuando el cliente no tiene perfil técnico y la obtención de feedback no puede garantizarse en plazos concretos.

6.4 Visión de futuro

El proyecto entregado constituye una base sólida sobre la que el centro puede crecer y evolucionar su presencia digital a lo largo del tiempo. A continuación se proponen las mejoras y ampliaciones más relevantes, organizadas según su prioridad y su viabilidad en el contexto real del centro.

Consolidación de la versión Next.js/Strapi

La mejora más significativa a corto plazo sería que el centro reconsiderara la decisión de no contratar hosting y dominio propio, y desplegara la versión técnica completa desarrollada con Next.js y Strapi como web definitiva. Esta versión ofrece una experiencia de usuario notablemente superior a la de NODES2, una gestión de contenidos mucho más flexible y potente, y una arquitectura preparada para crecer sin limitaciones técnicas. El coste de un hosting básico y un dominio propio es reducido, y la documentación elaborada durante el proyecto —especialmente el manual de uso de Strapi— garantiza que el equipo del centro podría gestionarla de forma autónoma con una formación inicial mínima. Esta sigue siendo la solución recomendada para el centro a medio plazo.

Ampliación de módulos de la Dynamic Zone

La arquitectura modular del sistema permite añadir nuevos tipos de módulo sin modificar la lógica existente. Entre los módulos que podrían incorporarse en el futuro destacan un módulo de noticias y blog con listado de entradas y vista de detalle, un módulo de calendario escolar con visualización mensual de eventos, un módulo de descarga de documentos para que las familias puedan acceder a autorizaciones, circulares y formularios directamente desde la web, y un módulo de galería de vídeos con soporte para colecciones organizadas por curso o etapa educativa. Cada uno de estos módulos seguiría el mismo patrón de implementación que los ya existentes, lo que garantiza su integración sin riesgo de romper el funcionamiento del sistema actual.

Multilingüismo

El centro imparte sus clases en catalán y tiene presencia en un entorno multilingüe. Strapi ofrece soporte nativo para la internacionalización mediante el plugin i18n, que permite gestionar versiones de cada contenido en distintos idiomas desde el mismo panel de administración. Implementar el multilingüismo permitiría al centro ofrecer el sitio en catalán, castellano e inglés, ampliando su alcance a familias de procedencias diversas y mejorando la accesibilidad del sitio para toda la comunidad educativa.

Mejora del rendimiento y el SEO

Aunque el sitio ya obtiene buenas puntuaciones en Lighthouse, existen margen de mejora en aspectos como la generación estática incremental (ISR) de páginas con contenido que cambia con poca frecuencia, la implementación de metadatos dinámicos por página configurables desde Strapi para mejorar el posicionamiento en buscadores, y la integración de datos estructurados en formato JSON-LD para que Google pueda indexar correctamente la información del centro como institución educativa.

Área privada para familias

Una ampliación de mayor envergadura a largo plazo sería la implementación de un área privada para familias, donde pudieran acceder a información personalizada como el menú del comedor, las circulares del curso, el calendario de reuniones o las notas de sus hijos. Esta funcionalidad requeriría implementar un sistema de autenticación de usuarios en Strapi y desarrollar las páginas y componentes correspondientes en el frontend, pero encajaría perfectamente en la arquitectura existente del proyecto.

Integración con ClickEdu

El centro utiliza ClickEdu como plataforma de gestión interna para la comunicación con las familias, el seguimiento académico y los trámites de secretaría. Una integración entre la web y ClickEdu permitiría mostrar en el sitio público información sincronizada en tiempo real —como el menú del comedor o los eventos del calendario— y facilitaría el acceso directo de las familias a la plataforma desde la propia web del centro, mejorando la coherencia de la experiencia digital del Lluís Millet en su conjunto.

Formación del equipo del centro

Más allá de las mejoras técnicas, una de las acciones más importantes para garantizar el éxito a largo plazo del proyecto sería realizar una sesión de formación presencial con el equipo del centro sobre el uso del CMS. Aunque el manual elaborado durante el proyecto cubre los aspectos principales de la gestión de contenidos, una formación práctica permitiría resolver dudas en tiempo real, adaptar la explicación al nivel y las necesidades concretas de cada miembro del equipo, y generar la confianza necesaria para que el centro se sienta capaz de gestionar el sitio de forma completamente autónoma. Esta confianza fue precisamente la que faltó en el momento de decidir si contratar o no el hosting, y una formación adecuada podría haber cambiado esa decisión.

7. Glosario

API REST — Interfaz de programación de aplicaciones que sigue los principios de la arquitectura REST (Representational State Transfer). Permite la comunicación entre sistemas mediante peticiones HTTP estándar (GET, POST, PUT, DELETE) e intercambio de datos en formato JSON. En este proyecto es el mecanismo mediante el cual Next.js obtiene los datos de Strapi.

App Router — Sistema de enrutamiento de Next.js introducido en la versión 13 que utiliza el sistema de ficheros como base para definir las rutas de la aplicación. Permite el uso de Server Components, layouts anidados y rutas dinámicas de forma nativa, y es el sistema utilizado en este proyecto para gestionar todas las páginas del sitio.

Breadcrumbs — Elemento de navegación secundaria que muestra al usuario la ruta jerárquica desde la página de inicio hasta la sección en la que se encuentra, permitiéndole navegar hacia niveles superiores con un solo clic. En español se conoce como migas de pan.

CLS (Cumulative Layout Shift) — Métrica de rendimiento web que mide la estabilidad visual de una página durante su carga. Un CLS elevado indica que los elementos de la página se desplazan de forma inesperada mientras se carga el contenido, lo que genera una experiencia de usuario deficiente. Se minimiza definiendo las dimensiones de las imágenes antes de que se carguen.

CMS (Content Management System) — Sistema de gestión de contenidos que permite crear, editar y publicar contenido digital desde un panel de administración sin necesidad de modificar el código fuente de la aplicación. En este proyecto se utiliza Strapi como CMS en modo headless.

CMS headless — Variante de CMS en la que el backend de gestión de contenidos está completamente desacoplado del frontend de presentación. El CMS expone los contenidos a través de una API y el frontend los consume de forma independiente, lo que permite utilizar cualquier tecnología en la capa de presentación.

Collection Type — Tipo de contenido de Strapi que permite crear múltiples registros del mismo tipo, como una colección de páginas o de artículos. Se diferencia del Single Type en que puede contener un número ilimitado de entradas.

Componente — En el contexto de React y Next.js, una función de JavaScript o TypeScript que recibe datos como props y devuelve elementos de interfaz de usuario. Los componentes son la unidad básica de construcción del frontend y pueden reutilizarse en distintas partes de la aplicación.

CTA (Call To Action) — Elemento de interfaz, normalmente un botón o un bloque destacado, diseñado para invitar al usuario a realizar una acción concreta, como contactar con el centro, descargar un documento o acceder a una sección específica del sitio.

Dynamic Zone — Funcionalidad de Strapi que permite definir un campo compuesto por una lista ordenada de componentes de distintos tipos. Cada entrada de la lista puede ser de un tipo diferente, lo que permite construir páginas con estructuras de contenido flexibles y variables desde el panel de administración.

Emoji — Símbolo gráfico estandarizado incluido en el estándar Unicode que puede utilizarse como texto en cualquier sistema sin necesidad de gestionar archivos de imagen. En este proyecto se utilizan como sistema de iconografía en los módulos gestionados desde Strapi.

Fallback — Mecanismo de respaldo que se activa cuando el proceso principal no produce el resultado esperado. En este proyecto, cuando el sistema no encuentra una página por su rutaCompleta, activa un fallback que intenta resolver la ruta utilizando únicamente el slug de la página.

Fetch — Función nativa de JavaScript y Next.js que permite realizar peticiones HTTP a una URL y obtener la respuesta en formato JSON u otros formatos. En este proyecto se utiliza para consultar la API REST de Strapi desde los Server Components de Next.js.

Figma — Herramienta de diseño de interfaces de usuario basada en navegador que permite crear prototipos interactivos, componentes reutilizables y guías de estilo de forma colaborativa. En este proyecto se utilizó para diseñar y validar el aspecto visual del sitio antes de iniciar el desarrollo.

Git — Sistema de control de versiones distribuido que permite registrar el historial de cambios de un proyecto de software, trabajar en ramas paralelas y revertir modificaciones. En este proyecto se utilizó junto con GitHub para gestionar el código fuente del frontend y el backend.

Google Fonts — Servicio gratuito de Google que proporciona fuentes tipográficas optimizadas para uso web. En este proyecto se utilizaron las familias Montserrat y Open Sans, obtenidas de este servicio sin coste adicional.

GitHub — Plataforma de alojamiento de repositorios Git en la nube que añade funcionalidades de colaboración, revisión de código y gestión de proyectos. En este proyecto se utilizó como repositorio central del código fuente del proyecto.

ISR (Incremental Static Regeneration) — Técnica de Next.js que combina la generación estática de páginas con la capacidad de regenerarlas de forma incremental en segundo plano cuando el contenido cambia, sin necesidad de redesplegar toda la aplicación.

LCP (Largest Contentful Paint) — Métrica de rendimiento web que mide el tiempo que tarda en renderizarse el elemento visual más grande de la página, normalmente una imagen o un bloque de texto destacado. Es uno de los indicadores principales de la experiencia de carga percibida por el usuario.

Lighthouse — Herramienta de auditoría web integrada en las herramientas de desarrollo de Google Chrome que analiza el rendimiento, la accesibilidad, las buenas prácticas y el SEO de un sitio web, proporcionando puntuaciones y recomendaciones de mejora.

Media Library — Gestor de archivos multimedia integrado en Strapi que permite subir, organizar y reutilizar imágenes y otros archivos desde el panel de administración. Todos los campos de tipo imagen de los módulos del proyecto apuntan a archivos almacenados en esta biblioteca.

Mobile-first — Enfoque de diseño y desarrollo web que consiste en diseñar primero para dispositivos móviles y luego adaptar el diseño a pantallas más grandes, en lugar del enfoque tradicional inverso. Garantiza que la experiencia en móvil sea óptima desde el inicio.

ModuleRenderer — Componente React desarrollado específicamente para este proyecto que recibe la lista de módulos de la Dynamic Zone de una página y renderiza el componente correspondiente a cada uno según su tipo, actuando como nexo central entre los datos de Strapi y los componentes visuales del frontend.

Monorepo — Estructura de repositorio en la que el código de varios proyectos o capas de un sistema —en este caso el frontend y el backend— se almacena en un único repositorio de Git, facilitando la gestión centralizada del código y el control de versiones compartido.

Next.js — Framework de React de código abierto desarrollado por Vercel que añade funcionalidades como el renderizado en servidor, la generación estática, el enrutamiento basado en ficheros y la optimización automática de imágenes. Es el framework utilizado para el desarrollo del frontend de este proyecto.

NODES2 (xtecagora) — Plataforma de gestión web proporcionada por la Generalitat de Catalunya para centros educativos concertados, basada en WordPress. No requiere infraestructura propia ni coste adicional para el centro y es la plataforma sobre la que se desarrolló la segunda versión del sitio web del proyecto.

Populate — Parámetro de la API REST de Strapi que indica qué relaciones y componentes anidados deben incluirse en la respuesta de una petición. Sin un populate correcto, los campos relacionados devuelven valores vacíos aunque estén configurados en el CMS.

Props — Abreviatura de properties. En React, son los datos que un componente padre pasa a un componente hijo para que los utilice en su renderizado. En este proyecto, cada módulo de la Dynamic Zone recibe sus datos de Strapi como props desde ModuleRenderer.

Render — Plataforma de alojamiento en la nube utilizada en este proyecto para desplegar el backend de Strapi en producción. Ofrece un plan gratuito con las prestaciones suficientes para alojar una instancia de Strapi en un proyecto de estas características.

Responsive / Responsivo — Característica de un sitio web que adapta automáticamente su diseño y distribución de contenidos al tamaño de la pantalla del dispositivo desde el que se accede, garantizando una experiencia de usuario óptima en móvil, tableta y escritorio.

RGPD (Reglamento General de Protección de Datos) — Normativa europea de obligado cumplimiento que regula el tratamiento de datos personales de los ciudadanos de la Unión Europea. Obliga a los sitios web a incluir política de privacidad, aviso legal y gestión de cookies, entre otros requisitos.

rutaCompleta — Campo personalizado definido en el content-type Pages de Strapi que almacena la ruta completa de cada página tal como aparece en la URL del sitio, por ejemplo /escola/historia-del-centre. Es el campo principal que utiliza el sistema de enrutamiento dinámico del frontend para resolver qué página mostrar en cada URL.

Server Component — Tipo de componente de Next.js que se ejecuta exclusivamente en el servidor y no envía JavaScript al navegador del cliente. Permite realizar operaciones costosas como llamadas a APIs o acceso a bases de datos directamente en el componente, mejorando el rendimiento y la seguridad de la aplicación.

shadcn/ui — Biblioteca de componentes de interfaz de código abierto basada en Tailwind CSS y Radix UI que proporciona componentes accesibles y personalizables como menús desplegables, acordeones y pestañas. En este proyecto se utilizó como base para algunos de los componentes de interfaz más complejos.

Single Type — Tipo de contenido de Strapi que solo permite un único registro, pensado para contenidos globales que aplican a todo el sitio, como la configuración general o los datos de contacto. En este proyecto se utiliza para el content-type Site-setting.

Slug — Identificador textual único de una página o recurso web, formado por palabras en minúsculas separadas por guiones, sin caracteres especiales. Se utiliza como parte de la URL y como mecanismo de fallback en el sistema de enrutamiento de este proyecto.

SSR (Server Side Rendering) — Técnica de renderizado web en la que el servidor genera el HTML completo de cada página antes de enviarlo al navegador del cliente. Mejora el rendimiento percibido y el SEO en comparación con las aplicaciones de página única que generan el HTML en el cliente.

Strapi — CMS headless de código abierto y autoalojable que permite definir estructuras de contenido personalizadas y exponerlas mediante una API REST. Es el backend utilizado en este proyecto para la gestión de contenidos del sitio web del Lluís Millet.

Tailwind CSS — Framework de CSS de código abierto basado en clases de utilidad que permite estilizar componentes directamente en el HTML sin escribir CSS personalizado. En este proyecto se utilizó como sistema de estilos principal del frontend.

TypeScript — Superconjunto tipado de JavaScript que añade un sistema de tipos estático al lenguaje. Permite detectar errores en tiempo de desarrollo antes de ejecutar el código y mejora la legibilidad y el mantenimiento del proyecto. Es el lenguaje utilizado en el frontend de este proyecto.

Vercel — Plataforma de despliegue en la nube especializada en aplicaciones Next.js, desarrollada por los mismos creadores del framework. En este proyecto se utilizó para desplegar el frontend del sitio web en producción, con integración automática con el repositorio de GitHub.

WCAG 2.1 (Web Content Accessibility Guidelines) — Estándar internacional de accesibilidad web desarrollado por el W3C que define los criterios que debe cumplir un sitio web para ser accesible a personas con distintas capacidades. El nivel AA de este estándar es el requisito mínimo exigible para sitios web institucionales y es el nivel que cumple el sitio desarrollado en este proyecto.

Cloudinary — Servicio de gestión y entrega de activos multimedia en la nube que permite almacenar, transformar y optimizar imágenes y vídeos de forma automática. En este proyecto se utiliza como proveedor de almacenamiento de imágenes para Strapi en producción, reemplazando el sistema de ficheros local del servidor, que no es persistente en la plataforma Render.

Session pooler — Mecanismo de gestión de conexiones a una base de datos que reutiliza conexiones existentes en lugar de abrir una nueva por cada petición. En este proyecto, Supabase proporciona la conexión a PostgreSQL mediante session pooler, lo que permite

que Strapi gestione la base de datos de forma estable sin agotar el límite de conexiones simultáneas del plan gratuito.

Supabase — Plataforma de código abierto que proporciona una base de datos PostgreSQL gestionada en la nube junto con herramientas adicionales como autenticación y almacenamiento. En este proyecto se utiliza exclusivamente como servicio de base de datos para Strapi, conectado mediante session pooler desde la instancia desplegada en Render.

UptimeRobot — Servicio de monitorización web que comprueba periódicamente la disponibilidad de un servicio o URL y notifica al administrador en caso de caída. En este proyecto se utiliza para enviar pings al backend de Strapi en Render cada cinco minutos, evitando que la instancia entre en estado de suspensión por inactividad, comportamiento habitual en los planes gratuitos de esa plataforma.

8. Bibliografía

Documentación oficial

Next.js Documentation. *Getting Started with the App Router*. Vercel. Disponible en: https://nextjs.org/docs

Strapi Documentation. *Developer Docs — REST API & Content Types*. Strapi Solutions. Disponible en: https://docs.strapi.io

Tailwind CSS Documentation. *Utility-First CSS Framework*. Tailwind Labs. Disponible en: https://tailwindcss.com/docs

shadcn/ui Documentation. *Beautifully designed components*. Disponible en: https://ui.shadcn.com/docs

Vercel Documentation. *Deploying Next.js Applications*. Vercel Inc. Disponible en: https://vercel.com/docs

Render Documentation. *Deploying a Node.js Application*. Render Services Inc. Disponible en: https://render.com/docs

Figma Help Center. *Getting started with prototyping*. Figma Inc. Disponible en: https://help.figma.com

Google Fonts. *Montserrat & Open Sans typefaces*. Google LLC. Disponible en: https://fonts.google.com

Accesibilidad y normativa

W3C Web Accessibility Initiative. *Web Content Accessibility Guidelines (WCAG) 2.1*. World Wide Web Consortium, 2018. Disponible en: https://www.w3.org/TR/WCAG21/

Agència Catalana de Protecció de Dades. *Guia pràctica per a webs i aplicacions*. Generalitat de Catalunya. Disponible en: https://apdcat.gencat.cat

Herramientas de análisis y rendimiento

Google. *Lighthouse — Web Performance Auditing Tool*. Google Developers. Disponible en: https://developer.chrome.com/docs/lighthouse

Google. *PageSpeed Insights*. Google LLC. Disponible en: https://pagespeed.web.dev

Centros educativos analizados

FEDAC Santa Coloma. Disponible en: https://www.fedac.cat/santa-coloma

Escola Manent. Disponible en: https://www.escolamanent.cat

CEIP Serra de Marina. Disponible en: https://agora.xtec.cat/ceipserrademarina

St Peter's School Barcelona. Disponible en: https://www.stpetersschool.eu

Escola Lluís Millet (web original analizada). Disponible en: https://blocs.xtec.cat/escolalluismillet

Plataformas educativas

Generalitat de Catalunya. *NODES2 — Plataforma de gestió web per a centres educatius*. Departament d'Educació. Disponible en: https://educacio.gencat.cat

Generalitat de Catalunya. *xtecagora — Servei de blocs educatius*. Departament d'Educació. Disponible en: https://agora.xtec.cat

Recursos de desarrollo y resolución de problemas

Stack Overflow. *Comunidad de preguntas y respuestas para desarrolladores*. Disponible en: https://stackoverflow.com

MDN Web Docs. *JavaScript, HTML y CSS Reference*. Mozilla Foundation. Disponible en: https://developer.mozilla.org

React Documentation. *Learn React*. Meta Open Source. Disponible en: https://react.dev/learn

TypeScript Documentation. *TypeScript Handbook*. Microsoft. Disponible en: https://www.typescriptlang.org/docs

GitHub Docs. *Managing repositories and version control*. GitHub Inc. Disponible en: https://docs.github.com

Inteligencia artificial

Claude se utilizó como herramienta de apoyo a lo largo de todo el proyecto en distintos ámbitos. En la fase de análisis y documentación, se empleó para estructurar y redactar secciones de la memoria del proyecto, elaborar la guía de estilo institucional y el documento de explicación del funcionamiento de Strapi orientado a usuarios no técnicos. En la fase de desarrollo, se utilizó para resolver dudas técnicas sobre Next.js, Strapi, TypeScript y Tailwind CSS, depurar errores de populate en la API de Strapi y revisar fragmentos de código. En la fase de presentación, se empleó para generar y iterar la presentación final del proyecto en formato PowerPoint, redactar el guión de exposición y preparar respuestas a posibles preguntas del tribunal. En todos los casos, el contenido generado fue revisado, contrastado y adaptado por el autor antes de su uso definitivo en el proyecto. Disponible en: https://claude.ai

Supabase Documentation. Connecting to your database — Session Pooler. Supabase Inc. Disponible en: https://supabase.com/docs

Cloudinary Documentation. Strapi Upload Provider. Cloudinary. Disponible en: https://cloudinary.com/documentation

UptimeRobot. Getting Started — HTTP(s) & Ping Monitors. UptimeRobot Ltd. Disponible en: https://uptimerobot.com/help

9. Anexos

A continuación se relacionan los documentos elaborados a lo largo del proyecto como entregables de cada fase, que complementan y amplían el contenido de esta memoria.

Anexo 1 — Anàlisi Lluís Millet Documento elaborado durante la Fase 1 del proyecto que recoge el análisis exhaustivo de la web existente del centro en blocs.xtec.cat, identificando sus problemas de diseño, accesibilidad, estructura y contenidos, y el estudio comparativo con cuatro centros educativos de referencia: FEDAC Santa Coloma, Escola Manent, CEIP Serra de Marina y St Peter's School Barcelona.

Anexo 2 — Proposta Organigrama Lluís Millet Documento elaborado al final de la Fase 1 que recoge las tres propuestas de arquitectura de navegación para la nueva web, junto con el formulario de feedback enviado a la dirección del centro para recopilar sus preferencias antes de definir la estructura definitiva. Incluye la propuesta seleccionada —la número 1— que sirvió como base para el desarrollo posterior del sitio.

Anexo 3 — Guia d'Estil Documento elaborado durante la Fase 2 que define la identidad visual institucional del centro: evolución del logotipo, paleta de colores en tonos verde oliva, tipografías Montserrat y Open Sans, criterios de iconografía y fotografía, y diseño de los componentes principales de interfaz. Sirvió como referencia visual para todo el desarrollo del frontend.

Anexo 4 — Document d'explicació de funcionament de Strapi Manual completo elaborado durante la Fase 3 que documenta el funcionamiento del CMS Strapi implementado en el proyecto: estructura de content-types, gestión de la Dynamic Zone, publicación de contenidos y administración de usuarios y permisos. Orientado a usuarios sin conocimientos técnicos para que el equipo del centro pueda gestionar el sitio de forma autónoma.

Anexo 5 — Repositorio del proyecto El código fuente completo del proyecto, incluyendo el frontend desarrollado con Next.js y el backend con Strapi, está disponible en el repositorio de GitHub del proyecto: https://github.com/JoseRibelles/Proyecto

Licencia