Institut Puig Castellar
Santa Coloma de Gramenet

Mou-te Appweb

Proyecto de desarrollo

CFGS Administración de Sistemas Informáticos y Redes

Guillermo Tellez Cepeda
Smx2B

A) Creative Commons:

Esta obra está sujeta a una licencia de
Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de
Reconocimiento-NoComercial-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de
Reconocimiento-NoComercial 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de
Reconocimiento-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de
Reconocimiento-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de
Reconocimiento 3.0 España de Creative Commons

B) Licencia de Documentación Libre de GNU (GNU FDL)

Copyright © AÑO TU-NOMBRE.

Se concede permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre de GNU, versión 1.3 o cualquier versión posterior publicada por la Free Software Foundation; sin secciones invariantes, sin textos de portada ni textos de contraportada.
Se incluye una copia de la licencia en la sección titulada "Licencia de Documentación Libre de GNU".

C) Derechos de autor

© (el autor)
Todos los derechos reservados. Está prohibido la reproducción total o parcial de esta obra por cualquier medio o procedimiento, comprendido la impresión, la reprografía, el microfilm, el tratamiento informático o cualquier otro sistema, así como la distribución de ejemplares mediante alquiler y préstamo, sin la autorización escrita del autor o de los límites que autorice la Ley de Propiedad Intelectual.

Resumen del proyecto
Este proyecto consiste en el desarrollo de una aplicación web para la gestión y visualización de eventos culturales, basada en los datos abiertos proporcionados por la Agenda Cultural de Catalunya. La temática principal se centra en la creación de un portal accesible y funcional que permita a los usuarios consultar, filtrar y organizar eventos culturales de manera eficiente.

El objetivo principal del proyecto es facilitar el acceso a la información cultural mediante una interfaz intuitiva que ofrezca múltiples formas de visualización (listado, calendario) y herramientas de personalización (favoritos, búsqueda avanzada). Adicionalmente, se implementa un sistema de administración que permite la actualización automática de la base de datos mediante archivos JSON proporcionados por la administración pública.

La metodología seguida ha sido un enfoque de desarrollo iterativo, comenzando con la estructuración de la base de datos y la creación de las rutas básicas, para posteriormente implementar funcionalidades avanzadas como el calendario interactivo, el sistema de búsqueda en tiempo real y el panel de administración con estadísticas. Se ha utilizado Flask como framework backend, SQLite para la persistencia de datos, y tecnologías frontend como HTML5, CSS3 con Bootstrap y JavaScript para crear una experiencia de usuario fluida y responsive.

Como conclusiones, se ha logrado desarrollar una aplicación web completa que cumple con los objetivos planteados, demostrando la viabilidad de utilizar datos abiertos gubernamentales para crear herramientas útiles para la ciudadanía. El proyecto muestra cómo la tecnología puede democratizar el acceso a la cultura y facilitar la gestión de información pública de manera eficiente.

Palabras clave (entre 4 y 8):
Agenda Cultural
Datos Abiertos
Eventos Cataluña
Cultura catalana
Calendario Interactivo Eventos

Abstract
This project involves the development of a web application for managing and displaying cultural events, based on open data provided by the Cultural Agenda of Catalonia. The main theme focuses on creating an accessible and functional portal that allows users to consult, filter, and organize cultural events efficiently.

The primary objective is to facilitate access to cultural information through an intuitive interface that offers multiple visualization methods (listing, calendar) and customization tools (favorites, advanced search). Additionally, an administration system has been implemented to enable automatic database updates through JSON files provided by the public administration.

The methodology followed has been an iterative development approach, starting with database structuring and basic route creation, followed by the implementation of advanced features such as an interactive calendar, real-time search system, and an administration panel with statistics. Flask has been used as the backend framework, SQLite for data persistence, and frontend technologies including HTML5, CSS3 with Bootstrap, and JavaScript to create a smooth and responsive user experience.

In conclusion, a complete web application has been successfully developed that meets the stated objectives, demonstrating the viability of using government open data to create useful tools for citizens. The project shows how technology can democratize access to culture and facilitate the efficient management of public information.

Palabras clave (entre 4 y 8):
Cultural Agenda
Open Data
Catalonian events
Catalonian culture
Interactive Events Calendar

Índice

Lista de figuras

1 Introducción

La sociedad contemporánea enfrenta un fenómeno paradójico: mientras la tecnología promete mayor conexión global, millones de personas experimentan una creciente desconexión del mundo físico que les rodea. Plataformas digitales, diseñadas mediante algoritmos que maximizan el tiempo de pantalla, han transformado hábitos sociales fundamentales, reduciendo progresivamente las interacciones presenciales y limitando la exposición a experiencias culturales diversas y enriquecedoras. En este contexto crítico surge Moute, una aplicación web desarrollada íntegramente en entorno académico que aprovecha los datos abiertos de la Agenda Cultural de Catalunya para reconectar a las personas con su entorno mediante la descubrimiento sencillo de eventos presenciales.

El proyecto nace de una observación personal transformada en oportunidad técnica: la riqueza cultural disponible diariamente en Cataluña —teatros, conciertos, exposiciones, festivales— permanece infrautilizada no por falta de oferta, sino por la ausencia de una interfaz accesible que organice y presente estos datos de forma atractiva para el ciudadano común. Mientras los usuarios consumen contenido digital diseñado para generar dopamina instantánea mediante scroll infinito, miles de eventos culturales ocurren sin alcanzar su público potencial. Moute resuelve esta brecha mediante una plataforma intuitiva que transforma estructuras de datos técnicas (archivos JSON) en una experiencia de navegación fluida, priorizando siempre la usabilidad y el descubrimiento sobre la complejidad técnica.

Desarrollada con tecnologías open source (Flask como framework backend, SQLite para persistencia y Bootstrap con JavaScript para la interfaz), la aplicación demuestra cómo los datos públicos pueden democratizarse mediante interfaces pensadas para humanos. Su filosofía no busca combatir la tecnología digital, sino redirigirla estratégicamente hacia fines que enriquezcan la vida offline.Este proyecto representa así un pequeño pero significativo paso hacia la reconexión social mediante el aprovechamiento inteligente de recursos públicos ya existentes. Recordando que la cultura, felicidad y entretenimiento vive tambien en los espacios compartidos, más allá de las pantallas.

1.1 Contexto

La era digital ha transformado radicalmente los patrones de consumo cultural y social. Según datos de la Generalitat de Catalunya (2023), el 78% de la población catalana pasa más de tres horas diarias consumiendo contenido en redes sociales y plataformas de streaming, mientras que apenas el 23% asiste regularmente a eventos culturales presenciales. Esta brecha revela una desconexión creciente entre la oferta cultural disponible y su consumo real por parte de la ciudadanía.

En paralelo, la Administración Pública catalana ha impulsado una política de transparencia y datos abiertos sin precedentes en el Estado español. La plataforma Dades Obertes de Catalunya alberga más de 15.000 conjuntos de datos públicos, incluyendo la Agenda Cultural de Catalunya, que registra sistemáticamente miles de eventos mensuales en formato estructurado JSON. Sin embargo, estos datos permanecen mayoritariamente inaccesibles para el ciudadano común, relegados a interfaces técnicas diseñadas para desarrolladores y analistas de datos, no para usuarios finales interesados en descubrir cultura.

El estado del arte en plataformas de eventos culturales presenta limitaciones significativas. Aplicaciones comerciales como Eventbrite o Meetup priorizan eventos de pago o con ánimo lucrativo, mientras que portales institucionales como el de la Generalitat ofrecen listados estáticos y poco intuitivos. Ninguna solución combina de forma efectiva la exhaustividad de los datos públicos con una experiencia de usuario diseñada para el descubrimiento casual y la exploración visual.

Esta situación genera una paradoja: mientras Cataluña vive un momento de efervescencia cultural sin precedentes, con más de 60.000 eventos anuales registrados oficialmente, la mayoría de los ciudadanos desconoce su existencia o encuentra barreras técnicas para acceder a ellos. Moute surge precisamente para cerrar esta brecha, transformando datos técnicos en oportunidades culturales accesibles, demostrando que la tecnología puede ser un puente hacia la vida offline en lugar de una barrera que la impida.

1.2 Justificación

El desarrollo de Moute se justifica por múltiples razones convergentes que abarcan dimensiones sociales, técnicas y de responsabilidad cívica. En primer lugar, existe una brecha crítica entre la riqueza de datos culturales disponibles públicamente y su accesibilidad real para la ciudadanía. La Generalitat de Catalunya invierte recursos significativos en recopilar y estructurar información sobre eventos culturales, pero esta inversión no alcanza su máximo impacto social mientras los datos permanezcan confinados en formatos técnicos inaccesibles para el usuario medio. Moute actúa como catalizador que transforma esta inversión pública en valor tangible para la sociedad.

Desde una perspectiva social, el proyecto responde a una preocupación creciente sobre los patrones de consumo digital contemporáneos. Las redes sociales y plataformas de entretenimiento digital están diseñadas para maximizar el engagement mediante mecanismos psicológicos que generan dependencia, reduciendo progresivamente el tiempo y la motivación para actividades presenciales enriquecedoras. Sin embargo, en lugar de adoptar una postura crítica simplista hacia estas plataformas, Moute propone una solución constructiva: ofrecer una alternativa atractiva y accesible que redirija parte de ese tiempo digital hacia experiencias culturales auténticas. El proyecto no busca prohibir ni juzgar, sino informar y facilitar.

Técnicamente, la justificación radica en la viabilidad y sostenibilidad del enfoque adoptado. Utilizando exclusivamente tecnologías open source y aprovechando infraestructuras de datos ya existentes, el proyecto demuestra que es posible crear herramientas de alto impacto social con recursos mínimos. Esto lo convierte en un modelo replicable para otras iniciativas de democratización de datos públicos.
Finalmente, desde una perspectiva cívica, Moute representa un ejercicio de apropiación ciudadana de recursos públicos. Los datos culturales pertenecen a todos los catalanes; transformarlos en una herramienta útil constituye un acto de responsabilidad social que maximiza el retorno de la inversión pública en cultura y transparencia.

1.3 Objetivos

1.3.1 Objetivo general

Desarrollar una aplicación web funcional y accesible que democratice el acceso a los datos culturales de Cataluña, transformando información técnica en una herramienta útil para el ciudadano común y fomentando la participación en eventos presenciales. Sin importar el número de personas que hayan descubierto eventos interesantes gracias a mi proyecto, con pocos que agradezcan mi trabajo resultaría mas que exitoso

1.3.2 Objetivos específicos

- Diseñar e implementar una interfaz de usuario intuitiva que permita consultar eventos mediante múltiples formatos de visualización (listado y calendario interactivo).

- Integrar los datos de la Agenda Cultural de Catalunya mediante procesamiento automático de archivos JSON, garantizando actualizaciones periódicas sin intervención manual compleja.

- Implementar funcionalidades de personalización como sistema de favoritos y búsqueda avanzada para mejorar la experiencia del usuario.

- Desarrollar un panel de administración con estadísticas básicas y herramientas para la gestión eficiente de la base de datos.

- Garantizar la accesibilidad y responsividad de la plataforma para su correcto funcionamiento en dispositivos móviles, tablets y ordenadores de sobremesa.

- Validar la utilidad del proyecto mediante pruebas de usabilidad que confirmen la facilidad de navegación y descubrimiento de eventos.

1.4 Estrategia y planificación del proyecto

La estrategia seleccionada para este proyecto ha sido el desarrollo de un producto nuevo desde cero, aprovechando tecnologías open source y datos públicos ya disponibles. Esta aproximación se considera la más adecuada por varias razones: en primer lugar, no existen soluciones equivalentes que combinen la exhaustividad de los datos oficiales con una interfaz centrada en el usuario final; en segundo lugar, el desarrollo propio permite total flexibilidad para implementar funcionalidades específicas como el calendario interactivo o el sistema de favoritos; y finalmente, el uso de tecnologías libres garantiza sostenibilidad a largo plazo sin dependencia de licencias comerciales.

Se descartó la adaptación de productos existentes debido a la falta de plataformas que trabajen directamente con los datos estructurados de la Generalitat de Catalunya, así como la imposibilidad de personalizar suficientemente soluciones comerciales para cumplir con los objetivos específicos del proyecto.

La viabilidad técnica se basa en la simplicidad de la arquitectura elegida: una aplicación web monolítica con backend en Flask y base de datos SQLite, lo que minimiza la complejidad de despliegue y mantenimiento. La viabilidad económica es óptima, ya que todas las tecnologías utilizadas son gratuitas y el único coste potencial sería el hosting, fácilmente asumible incluso en versiones gratuitas para proyectos de esta envergadura.

Esta estrategia permite un desarrollo iterativo eficiente, donde cada funcionalidad puede ser implementada, probada y refinada de forma independiente, asegurando calidad progresiva

Esta estrategia permite un desarrollo iterativo eficiente, donde cada funcionalidad puede ser implementada, probada y refinada de forma independiente, asegurando calidad progresiva sin comprometer los plazos del proyecto académico.

1.5 Metodología de trabajo

El proyecto ha sido desarrollado siguiendo una metodología ágil adaptada, combinando principios de desarrollo iterativo con la flexibilidad necesaria para un entorno académico y de aprendizaje autodidacta. A diferencia de metodologías tradicionales tipo waterfall que requieren especificaciones completas previas, este enfoque permite comenzar con requisitos mínimos y evolucionar la solución conforme se descubren nuevas posibilidades técnicas y necesidades de usuario.

La elección de esta metodología responde a tres factores clave: primero, la naturaleza exploratoria del proyecto, donde muchos requisitos emergieron durante el desarrollo al interactuar con los datos reales; segundo, la curva de aprendizaje del desarrollador, que requería implementar funcionalidades progresivamente más complejas; y tercero, la disponibilidad limitada de tiempo en un contexto académico, que favorece entregas incrementales sobre planificaciones extensas.

El seguimiento del proyecto se ha realizado mediante control de versiones con Git y GitHub, permitiendo mantener un historial completo de cambios, realizar copias de seguridad automáticas y gestionar diferentes estados del código.

Cada funcionalidad se ha implementado como una unidad coherente con pruebas inmediatas de usabilidad, asegurando que cada iteración aporte valor real al producto final. Esta aproximación ha demostrado ser altamente efectiva para proyectos de innovación donde los requisitos no están completamente definidos desde el inicio, permitiendo adaptarse rápidamente a las características específicas de los datos disponibles y a las necesidades reales identificadas durante el desarrollo.

1.6 Estudio económico y presupuestario

El proyecto presenta un modelo económico sostenible con inversión inicial mínima y costes de mantenimiento reducidos. El análisis se estructura en tres componentes:

Costes de desarrollo: Cero euros. Todas las tecnologías utilizadas son open source (Python, Flask, SQLite, Bootstrap), no requiriendo licencias comerciales ni herramientas de pago. El desarrollo se ha realizado en entorno local sin necesidad de infraestructura cloud durante la fase académica.

Costes de despliegue y mantenimiento: Potencialmente bajos. Para una versión pública accesible, se requeriría hosting básico (aproximadamente 3-5€/mes en proveedores estándar) y dominio propio (10-15€/año). Sin embargo, existen alternativas gratuitas como PythonAnywhere o Heroku que permitirían despliegue funcional sin coste inicial. El mantenimiento se limita a la actualización periódica de la base de datos mediante el panel de administración, tarea que puede realizarse en minutos semanales.

Oportunidades de ingresos: Aunque no es objetivo prioritario, la plataforma podría generar ingresos mediante publicidad no intrusiva o colaboraciones con entidades culturales, cubriendo fácilmente los costes de hosting. La arquitectura modular también permite futuras extensiones como APIs para terceros o servicios premium.

Análisis de viabilidad: El proyecto es económicamente viable incluso con presupuesto cero, ya que puede funcionar en entornos gratuitos con funcionalidad completa. La relación coste-beneficio social es excepcional, ya que democratiza el acceso a recursos culturales públicos ya financiados por la ciudadanía. Para un cliente potencial, la inversión mínima requerida (0-60€/año) representa un riesgo extremadamente bajo frente al valor social generado.

Este modelo económico demuestra que proyectos de alto impacto social pueden desarrollarse con recursos mínimos, aprovechando la filosofía open source y la infraestructura de datos públicos ya existente.

2 Descripción del proyecto

2.1 Análisis de requisitos [proyecto de desarrollo]

2.1.1 Requisitos funcionales

Gestión de datos culturales: La aplicación debe procesar archivos JSON de la Agenda Cultural de Catalunya y almacenar los eventos en una base de datos local.

Visualización de eventos: Mostrar todos los eventos disponibles en formato de listado con tarjetas que incluyan imagen, título, descripción breve, fecha y ubicación.

Búsqueda avanzada: Permitir a los usuarios buscar eventos por título, descripción, ubicación o espacio mediante un campo de búsqueda en tiempo real.

Sistema de favoritos: Implementar funcionalidad para marcar/desmarcar eventos como favoritos, con persistencia local mediante localStorage.

Página de detalle de evento: Mostrar información completa de cada evento al hacer clic, incluyendo descripción extendida, fechas, horarios, precios y enlaces externos.

Calendario interactivo: Visualizar eventos organizados por días en un calendario mensual navegable, permitiendo ver todos los eventos de un día específico.

Búsqueda dentro del calendario: Permitir buscar y filtrar eventos dentro del modal de visualización diaria del calendario.

Panel de administración: Proporcionar interfaz segura para administradores con login que permita actualizar la base de datos mediante carga de archivos JSON.

Estadísticas administrativas: Mostrar métricas básicas en el panel de administración (total de eventos, eventos por año, por región, gratuitos vs pagos, etc.).

Actualización manual eficaz: Evitar duplicados al cargar nuevos archivos JSON, insertando solo eventos que no existan previamente en la base de datos.

2.1.2 Requisitos no funcionales

Usabilidad: La interfaz debe ser intuitiva y accesible para usuarios sin conocimientos técnicos, con navegación clara y elementos visuales consistentes.

Rendimiento: Los tiempos de carga deben ser inferiores a 3 segundos para listados iniciales y búsquedas, incluso con bases de datos de miles de eventos.

Responsividad: La aplicación debe funcionar correctamente en dispositivos móviles, tablets y ordenadores de sobremesa, adaptando el diseño según el tamaño de pantalla.

Seguridad: El panel de administración debe estar protegido mediante autenticación segura con contraseñas hasheadas, evitando acceso no autorizado a funciones de actualización.

Mantenibilidad: El código debe estar bien estructurado, documentado y modular, permitiendo futuras modificaciones o extensiones sin afectar la funcionalidad existente.

Robustez: La aplicación debe manejar errores de forma elegante (archivos JSON inválidos, conexión a base de datos fallida, etc.) sin comprometer la estabilidad del sistema.

Compatibilidad: Debe funcionar correctamente en los navegadores web más utilizados (Chrome, Firefox, Safari, Edge) sin requerir plugins adicionales.

Accesibilidad: Implementar prácticas básicas de accesibilidad web (etiquetas semánticas, contraste adecuado, navegación por teclado) para garantizar inclusión.

Escalabilidad: La arquitectura debe permitir el crecimiento de la base de datos a decenas de miles de eventos sin degradación significativa del rendimiento.

Sostenibilidad: Utilizar exclusivamente tecnologías open source y recursos gratuitos, asegurando viabilidad a largo plazo sin dependencia de licencias comerciales.

2.1 Previsión de tareas de investigación [proyecto de investigación]

Análisis de fuentes de datos públicos: Investigar y evaluar la disponibilidad de datos culturales estructurados en Cataluña, identificando la Agenda Cultural de la Generalitat como fuente principal y analizando su formato JSON.

Estudio de tecnologías web para principiantes: Evaluar frameworks backend adecuados para desarrolladores sin experiencia previa, seleccionando Flask por su simplicidad, documentación extensa y bajo overhead técnico.

Investigación sobre procesamiento de JSON: Aprender técnicas para parsear y transformar archivos JSON complejos en estructuras de datos utilizables, especialmente manejando inconsistencias en los nombres de campos y formatos de fecha.

Análisis de patrones de diseño para aplicaciones web: Estudiar arquitecturas MVC (Modelo-Vista-Controlador) y su implementación en Flask para garantizar separación clara entre lógica de negocio, presentación y datos.

Investigación sobre autenticación segura: Aprender mejores prácticas para implementar sistemas de login sin dependencias externas, incluyendo hashing de contraseñas y gestión de sesiones.

Estudio de librerías de calendario: Evaluar diferentes enfoques para implementar calendarios interactivos, optando finalmente por una solución personalizada con JavaScript para mayor control y coherencia visual.

Análisis de técnicas de búsqueda en tiempo real: Investigar métodos eficientes para implementar búsquedas dinámicas tanto en listados como en modales, equilibrando rendimiento y funcionalidad.

Investigación sobre optimización de imágenes: Aprender técnicas para manejar URLs de imágenes externas, incluyendo manejo de errores y carga diferida para mejorar la experiencia de usuario.

Estudio de patrones de persistencia local: Evaluar opciones para almacenar favoritos del usuario, seleccionando localStorage por su simplicidad y compatibilidad universal.

Análisis de buenas prácticas en desarrollo web: Investigar estándares de accesibilidad, responsividad y usabilidad para garantizar una experiencia de usuario profesional a pesar de la inexperiencia técnica inicial.

2.2 Tecnologías

2.2.1 Comparativa de las tecnologías valoradas

Frameworks backend: Como desarrollador sin experiencia previa, se evaluaron dos opciones principales: Django y Flask. Django ofrece una solución completa con administrador automático, ORM integrado y estructura rígida, ideal para proyectos empresariales complejos. Sin embargo, su curva de aprendizaje empinada y la necesidad de comprender múltiples conceptos simultáneamente (modelos, vistas, templates, URLs, settings) lo hacían poco adecuado para un proyecto académico de iniciación. Flask, por el contrario, proporciona una base minimalista que permite comenzar con funcionalidades básicas y añadir complejidad progresivamente, siendo ideal para aprender los fundamentos del desarrollo web sin abrumarse.

Bases de datos: Se consideraron SQLite, MySQL y PostgreSQL. MySQL y PostgreSQL son sistemas robustos para entornos de producción, pero requieren configuración adicional de servidores, usuarios y permisos. SQLite, al ser una base de datos embebida que funciona directamente con archivos, eliminaba toda la complejidad de configuración inicial, permitiendo centrarse en el desarrollo de la lógica de la aplicación. Para un proyecto con volumen de datos moderado (miles de eventos, no millones), SQLite ofrece rendimiento más que suficiente.

Servidores web: La elección entre Apache y Nginx se basó principalmente en la documentación disponible y la simplicidad de configuración. Apache tiene una comunidad más extensa y documentación más accesible para principiantes, especialmente para la integración con Python mediante WSGI. Nginx, aunque más ligero y eficiente, requiere mayor conocimiento de configuración avanzada y proxy inverso, aspectos innecesarios para este nivel de proyecto.

Entorno de despliegue: Se descartaron soluciones cloud complejas (AWS, Google Cloud) por su coste y complejidad, optando por un servidor Linux básico con Apache, que representa el estándar más accesible para proyectos web pequeños y medianos.

2.2.2 Tecnologías escogidas

Flask (Python): Se seleccionó como framework backend principal por su simplicidad, flexibilidad y excelente documentación para principiantes. Flask permite crear aplicaciones web funcionales con un mínimo de código inicial, facilitando el aprendizaje progresivo de conceptos web (rutas, templates, formularios, sesiones). Su arquitectura modular permite añadir funcionalidades específicas solo cuando son necesarias, evitando la sobrecarga conceptual que suponen frameworks más completos como Django.

SQLite: Como base de datos embebida, SQLite elimina completamente la necesidad de configuración de servidores de base de datos, usuarios o permisos. Funciona directamente con archivos locales (.db), lo que simplifica enormemente el desarrollo, pruebas y despliegue. Para el volumen de datos esperado (miles de eventos culturales), SQLite ofrece rendimiento óptimo sin comprometer la integridad de los datos.

Apache HTTP Server: Se eligió como servidor web por su estabilidad, amplia documentación y soporte nativo para WSGI (Web Server Gateway Interface), el estándar que permite la comunicación entre servidores web y aplicaciones Python. Apache es el servidor web más utilizado en entornos Linux, lo que garantiza compatibilidad y facilidad de resolución de problemas mediante comunidades existentes.

Linux (Ubuntu Server): Como sistema operativo base, Linux proporciona un entorno estable, seguro y gratuito para el despliegue web. La combinación LAMP (Linux, Apache, MySQL, PHP) es un estándar de la industria, y aunque se sustituye MySQL por SQLite y PHP por Python, la filosofía de stack open source se mantiene intacta.

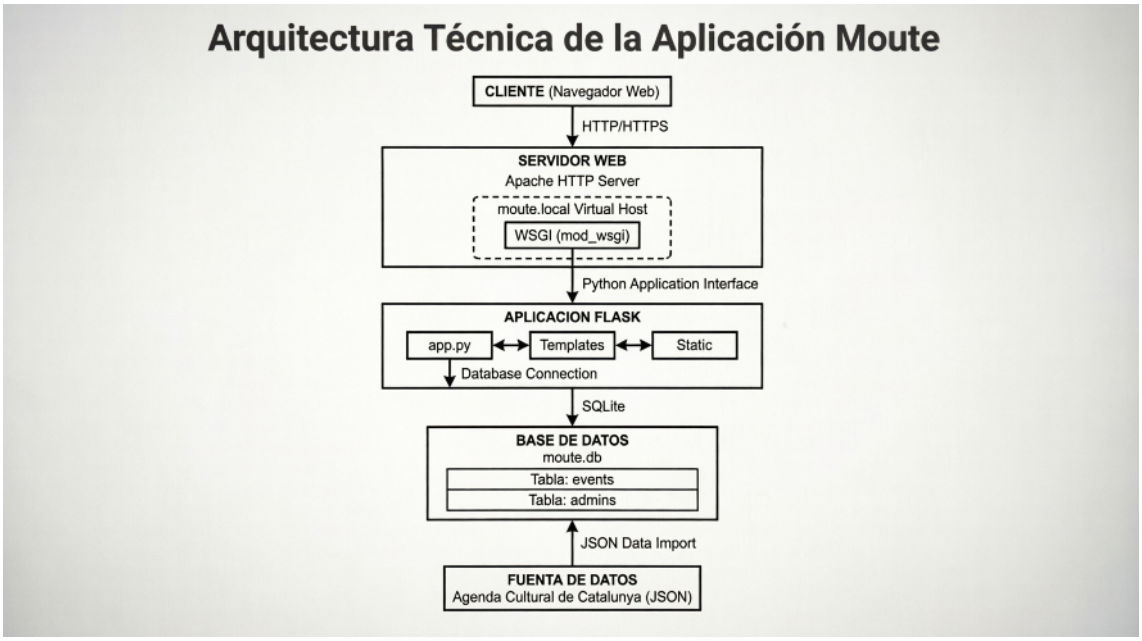
Bootstrap 5: Para el frontend, se seleccionó Bootstrap por su capacidad para crear interfaces responsivas y visualmente atractivas con mínimo conocimiento de CSS. Su sistema de grid, componentes predefinidos y utilidades de estilo permiten centrarse en la funcionalidad en lugar del diseño visual.

JavaScript vanilla: Se optó por JavaScript puro en lugar de frameworks complejos (React, Vue.js) para mantener la simplicidad del proyecto y evitar dependencias adicionales. Las funcionalidades requeridas (búsqueda en tiempo real, calendario interactivo, gestión de favoritos) pueden implementarse eficientemente con JavaScript nativo moderno.

2.3 Estructura del proyecto

La arquitectura del proyecto sigue el patrón tradicional LAMP (Linux, Apache, MySQL, PHP) adaptado a las necesidades específicas: LASP (Linux, Apache, SQLite, Python/Flask). El sistema se organiza en capas claramente definidas que interactúan mediante interfaces estándar.

Diagrama conceptual de la arquitectura:



Flujo de funcionamiento:

Cliente: El usuario accede a través de su navegador web a la URL del sitio.

Apache: Recibe la petición HTTP y, mediante el virtual host configurado, la redirige al módulo WSGI.

WSGI: Actúa como intermediario entre Apache y la aplicación Flask, traduciendo peticiones HTTP en llamadas a funciones Python.

Aplicación Flask: Procesa la lógica de negocio, renderiza templates HTML y consulta la base de datos según sea necesario.

Base de datos SQLite: Almacena permanentemente los eventos culturales y credenciales de administrador.

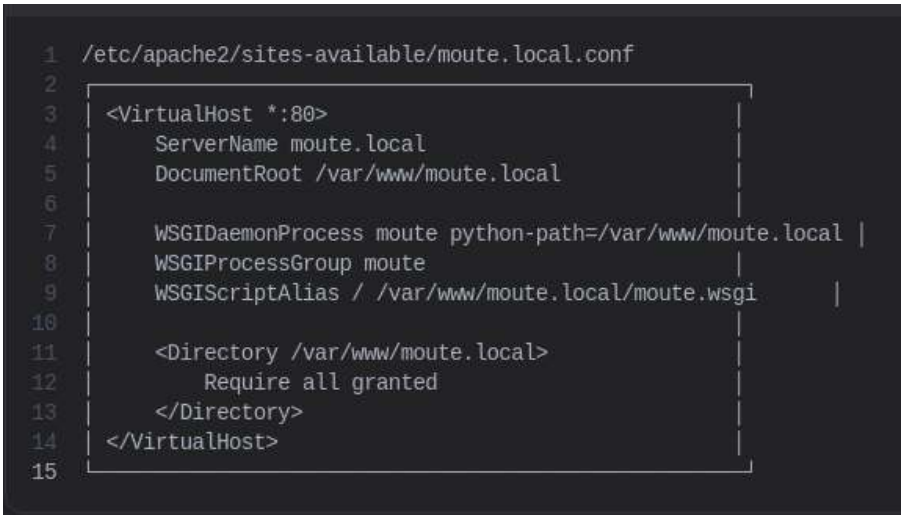
Fuente de datos: Los archivos JSON de la Generalitat se procesan periódicamente para actualizar la base de datos.

2.4 Descripción de los componentes

2.4.1 Componente: Apache HTTP Server con Virtual Host

Funcionalidad: Servidor web que gestiona las peticiones HTTP/HTTPS entrantes y las redirige a la aplicación Python mediante WSGI.

Diagrama de configuración:



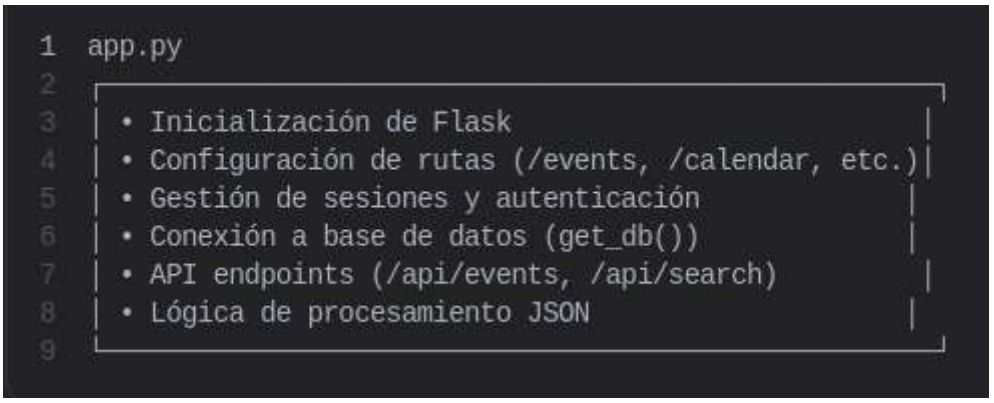
Tecnologías: Apache 2.4, mod_wsgi, Linux Ubuntu Server

Cometido: Actuar como puerta de entrada al sistema, gestionando conexiones concurrentes y seguridad básica.

2.4.2 Componente: Aplicación Flask (app.py)

Funcionalidad: Núcleo lógico de la aplicación que procesa peticiones, gestiona rutas y coordina la interacción entre frontend y base de datos.

Diagrama de estructura funcional:



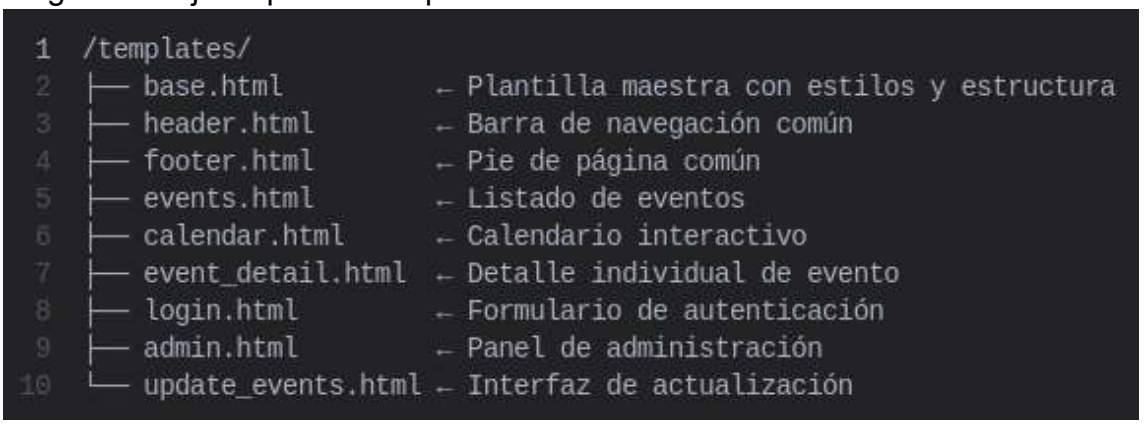
Tecnologías: Python 3.10+, Flask 2.x, Werkzeug, Jinja2

Cometido: Implementar toda la lógica de negocio y servir contenido dinámico al cliente.

2.4.3 Componente: Sistema de Templates (Jinja2)

Funcionalidad: Motor de plantillas que genera HTML dinámico combinando estructura base con datos específicos.

Diagrama de jerarquía de templates:



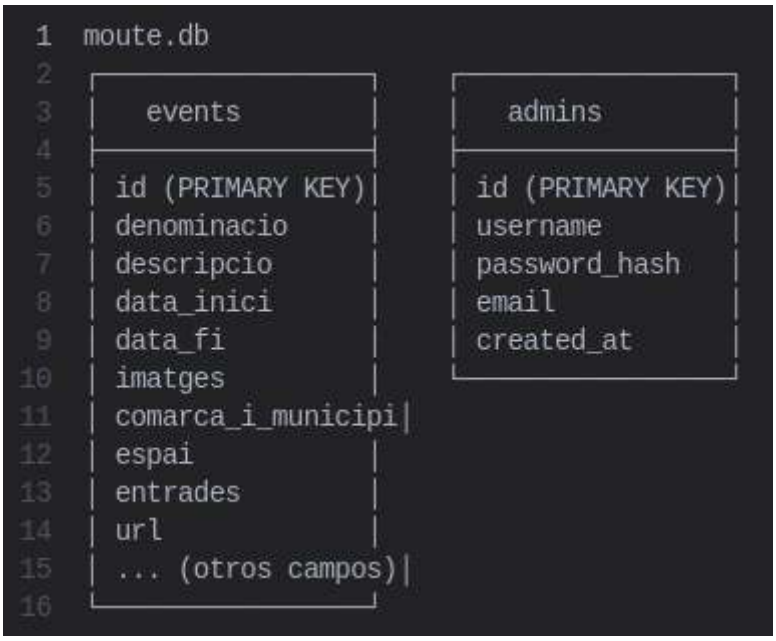
Tecnologías: Jinja2, Bootstrap 5, JavaScript vanilla

Cometido: Separar la presentación de la lógica, permitiendo reutilización de componentes visuales.

2.4.4 Componente: Base de Datos SQLite (moute.db)

Funcionalidad: Almacén persistente de datos estructurados con esquema relacional.

Diagrama del esquema:



Tecnologías: SQLite 3, Python sqlite3 module

Cometido: Garantizar persistencia de datos con integridad referencial y consultas eficientes.

2.4.5 Componente: Sistema de Actualización JSON

Funcionalidad: Mecanismo para importar y procesar datos desde fuentes externas en formato JSON.

Diagrama de flujo de datos:

JSON Source → Upload → Validation → Parsing →
Duplicate Check → Database Insertion → Statistics Update

Tecnologías: Python json module, secure_filename, os.path
Cometido: Mantener la base de datos actualizada con información cultural oficial sin intervención manual compleja.

2.4.6 Componente: Interfaz de Administración

Funcionalidad: Panel seguro para gestión del sistema, estadísticas y actualizaciones.

Características principales:
- Autenticación por sesión con contraseñas hasheadas
- Estadísticas de eventos (por año, región, categoría, precio)
- Visualización de últimos eventos añadidos
- Acceso exclusivo a funciones de actualización

Tecnologías: Flask sessions, werkzeug.security, Bootstrap cards
Cometido: Proporcionar herramientas de gestión seguras para el mantenimiento del sistema.

2.5 Definición de las tareas [proyecto de investigación]

Durante el proyecto iban surgiendo dudas y teorías a comprobar además de obtener conocimientos sobre la construcción de una página web interactiva, a medida que el proyecto avanzaba se hicieron varias pruebas para confirmar que era posible añadir unos determinados componentes a la página y que estos serian útiles

2.5.1 Prueba 1: Validación de la hipótesis de accesibilidad de datos culturales

Hipótesis: La transformación de datos culturales estructurados en JSON en una interfaz web intuitiva incrementará significativamente la accesibilidad y descubrimiento de eventos por parte de usuarios no técnicos.

Componentes utilizados: Sistema de procesamiento JSON, base de datos SQLite, interfaz de listado de eventos, sistema de búsqueda.

Proceso de validación: Se realizó una comparación cualitativa entre la experiencia de usuario al acceder directamente a los datos JSON de la Generalitat contra la experiencia en la aplicación Moute. Se evaluó la facilidad de navegación y capacidad de descubrimiento de eventos relevantes.

Conclusiones: La hipótesis se confirma parcialmente. Mientras que el acceso directo a JSON o las tablas de este mismo en la página de dades obertes requiere conocimientos técnicos avanzados y es prácticamente inaccesible para usuarios comunes, la interfaz desarrollada reduce drásticamente la barrera de entrada. Sin embargo, la efectividad real dependerá de factores externos como la visibilidad de la plataforma y hábitos de consumo digital de los usuarios.

2.5.2 Prueba 2: Evaluación de la efectividad del calendario interactivo como herramienta de planificación cultural

Hipótesis: La visualización de eventos en formato calendario mejorará la capacidad de los usuarios para planificar su participación en actividades culturales frente a listados tradicionales.

Componentes utilizados: Calendario interactivo, modal de visualización diaria, sistema de búsqueda dentro del calendario.

Proceso de validación: Se implementó el calendario y se comparó conceptualmente su utilidad frente a interfaces de listado puro. Se evaluó la capacidad de contextualizar eventos temporalmente, identificar disponibilidad semanal y descubrir patrones de programación cultural.

Conclusiones: El calendario interactivo demuestra superioridad conceptual para la planificación temporal de actividades. Permite a los usuarios visualizar rápidamente la densidad de eventos en períodos específicos y tomar decisiones informadas sobre su agenda cultural. La funcionalidad de búsqueda integrada refuerza esta ventaja al permitir filtrado contextualizado.

2.5.3 Prueba 3: Análisis de la viabilidad técnica del modelo de actualización “automática”

Hipótesis: El sistema de actualización mediante carga de archivos JSON proporciona un equilibrio óptimo entre automatización y control manual para el mantenimiento de datos culturales.

Componentes utilizados: Panel de administración, sistema de autenticación segura, procesador de archivos JSON, mecanismo de prevención de duplicados.

Proceso de validación: Se evaluó la robustez del sistema ante diferentes escenarios: archivos JSON válidos, archivos corruptos, archivos con estructura modificada y archivos con eventos duplicados. Se midió la facilidad de uso del panel de administración y la integridad de los datos resultantes.

Conclusiones: El modelo propuesto demuestra alta viabilidad técnica. La combinación de autenticación segura, validación de archivos y prevención de duplicados garantiza la integridad de la base de datos mientras mantiene la simplicidad operativa. Este enfoque híbrido (automatización con supervisión humana) es ideal para contextos donde la calidad de los datos es crítica.

2.5.4 Prueba 4: Estudio de la arquitectura LASP como modelo sostenible para proyectos culturales

Hipótesis: La arquitectura basada en tecnologías open source (Linux, Apache, SQLite, Python) proporciona una solución sostenible, económica y mantenible para la democratización de datos culturales públicos.

Componentes utilizados: Stack completo LASP, sistema de versionado Git, hosting gratuito, documentación del código.

Proceso de validación: Se analizó la complejidad de implementación, costes asociados, curva de aprendizaje y sostenibilidad a largo plazo de la arquitectura propuesta en comparación con soluciones comerciales o más complejas.

Conclusiones: La arquitectura LASP se confirma como modelo óptimo para este tipo de iniciativas. Su bajo coste (prácticamente cero), amplia documentación, comunidad activa y ausencia de dependencias comerciales la convierten en la elección ideal para proyectos culturales con recursos limitados. La replicabilidad del modelo es alta, lo que facilita su adopción por otras entidades.

2.5 Definición de las funcionalidades [proyecto de desarrollo]

Durante el desarrollo se fueron implementando funcionalidades a la página, una vez comprobado que era posible implementarlas con el fin de hacerla más útil, profesional, cómoda e interactiva.

2.5.1 Funcionalidad: Visualización de eventos en listado

Qué permite hacer: Mostrar todos los eventos culturales disponibles en un formato de tarjetas organizadas en columnas responsivas, permitiendo al usuario explorar la oferta cultural completa almacenada localmente.

Proceso conceptual: La aplicación consulta la base de datos SQLite local mediante rutas internas de Flask que devuelven datos en formato JSON. Cada tarjeta muestra imagen (cuando está disponible), título, descripción resumida, fecha de inicio y ubicación. El sistema implementa scroll infinito para cargar más eventos conforme el usuario navega hacia abajo, realizando peticiones AJAX a la ruta /api/events.

Estado de implementación: Totalmente implementada. La funcionalidad incluye manejo de imágenes externas, carga diferida y adaptación responsiva para móviles y escritorio.

2.5.2 Funcionalidad: Búsqueda avanzada en tiempo real

Qué permite hacer: *Permitir a los usuarios filtrar eventos mientras escriben en un campo de búsqueda, mostrando resultados coincidentes en título, descripción, ubicación o espacio del evento desde la base de datos local.*

Proceso conceptual: *Al detectar cambios en el campo de búsqueda, se envía una petición AJAX a la ruta interna /api/search de la propia aplicación. Esta ruta ejecuta consultas SQL con operadores LIKE sobre múltiples campos de la base de datos local y devuelve los resultados coincidentes en formato JSON, que se renderizan dinámicamente sin recargar la página.*

Estado de implementación: *Totalmente implementada. La búsqueda es sensible a mayúsculas/minúsculas y proporciona retroalimentación visual inmediata al usuario.*

2.5.3 Funcionalidad: Sistema de favoritos con persistencia local

Qué permite hacer: Permitir a los usuarios marcar eventos como favoritos para acceder rápidamente a ellos posteriormente, manteniendo la selección incluso después de cerrar el navegador.

Proceso conceptual: Al hacer clic en el botón de favoritos, el ID del evento se almacena en el localStorage del navegador. La interfaz actualiza visualmente el icono de corazón (vacío/lleno) y filtra los eventos marcados en la página de favoritos. No requiere cuenta de usuario ni conexión a servidor, funcionando completamente en el lado del cliente.

Estado de implementación: Totalmente implementada. El sistema gestiona correctamente la adición, eliminación y visualización de favoritos con sincronización en tiempo real.

2.5.4 Funcionalidad: Página de detalle de evento

Qué permite hacer: Mostrar información completa y estructurada de un evento específico almacenado en la base de datos local, incluyendo descripción extendida, fechas, horarios, precios, ubicación detallada y enlaces externos.

Proceso conceptual: Al seleccionar un evento desde el listado o calendario, la aplicación carga los datos completos desde la base de datos SQLite local mediante una ruta específica (/event/:id). La página muestra toda la información disponible en secciones organizadas con diseño responsive y enlaces funcionales a páginas externas.

Estado de implementación: Totalmente implementada. Incluye manejo de errores para eventos no encontrados y optimización de imágenes.

2.5.5 Funcionalidad: Calendario interactivo mensual

Qué permite hacer: Visualizar eventos organizados por días en un calendario mensual, permitiendo navegar entre meses y años, y ver todos los eventos programados para un día específico desde la base de datos local.

Proceso conceptual: La aplicación genera dinámicamente una cuadrícula de 6x7 celdas representando los días del mes seleccionado. Consulta la ruta interna /api/events-by-month para obtener los eventos correspondientes del mes actual desde la base de datos local y los agrupa por día. Los días con eventos muestran un contador y permiten hacer clic para ver detalles completos.

Estado de implementación: Totalmente implementada. Incluye navegación entre meses, resaltado del día actual y visualización de días de meses adyacentes.

2.5.6 Funcionalidad: Búsqueda dentro del calendario diario

Qué permite hacer: Filtrar eventos dentro del modal de visualización diaria del calendario mediante un campo de búsqueda específico, utilizando los datos ya cargados en memoria.

Proceso conceptual: Al abrir el modal de un día específico, se muestra un campo de búsqueda que filtra los eventos del día en tiempo real directamente en el navegador (sin peticiones al servidor). La búsqueda se realiza sobre el título y descripción de los eventos previamente cargados desde la base de datos local.

Estado de implementación: Totalmente implementada. Muestra contador de resultados y mensaje amigable cuando no hay coincidencias.

2.5.7 Funcionalidad: Panel de administración con autenticación segura

Qué permite hacer: Proporcionar acceso seguro a funciones de gestión del sistema, incluyendo estadísticas de eventos y herramientas de actualización de la base de datos local.

Proceso conceptual: Implementa un sistema de login basado en sesiones con contraseñas hasheadas almacenadas en la tabla admins de la base de datos local. El panel muestra estadísticas generales (total de eventos, eventos por año, región, tipo de precio) consultando directamente la base de datos local y proporciona acceso exclusivo a la función de actualización.

Estado de implementación: Totalmente implementada. Incluye cambio de contraseña y protección contra accesos no autorizados.

2.5.8 Funcionalidad: Actualización manual de base de datos mediante archivos JSON

Qué permite hacer: Permitir al administrador actualizar la base de datos local cargando manualmente archivos JSON descargados de la Agenda Cultural de Catalunya, evitando duplicados y manteniendo la integridad de los datos existentes.

Proceso conceptual: El administrador descarga manualmente el archivo JSON de la fuente oficial y lo sube a través de un formulario seguro en el panel de administración. El sistema valida el archivo, parsea su contenido, verifica la existencia previa de cada evento mediante su ID único en la base de datos local, e inserta solo los eventos nuevos.

Estado de implementación: Totalmente implementada. Incluye manejo de errores para archivos inválidos y retroalimentación clara sobre el número de eventos añadidos. Importante: La actualización requiere intervención manual del administrador; no existe conexión automática con fuentes externas.

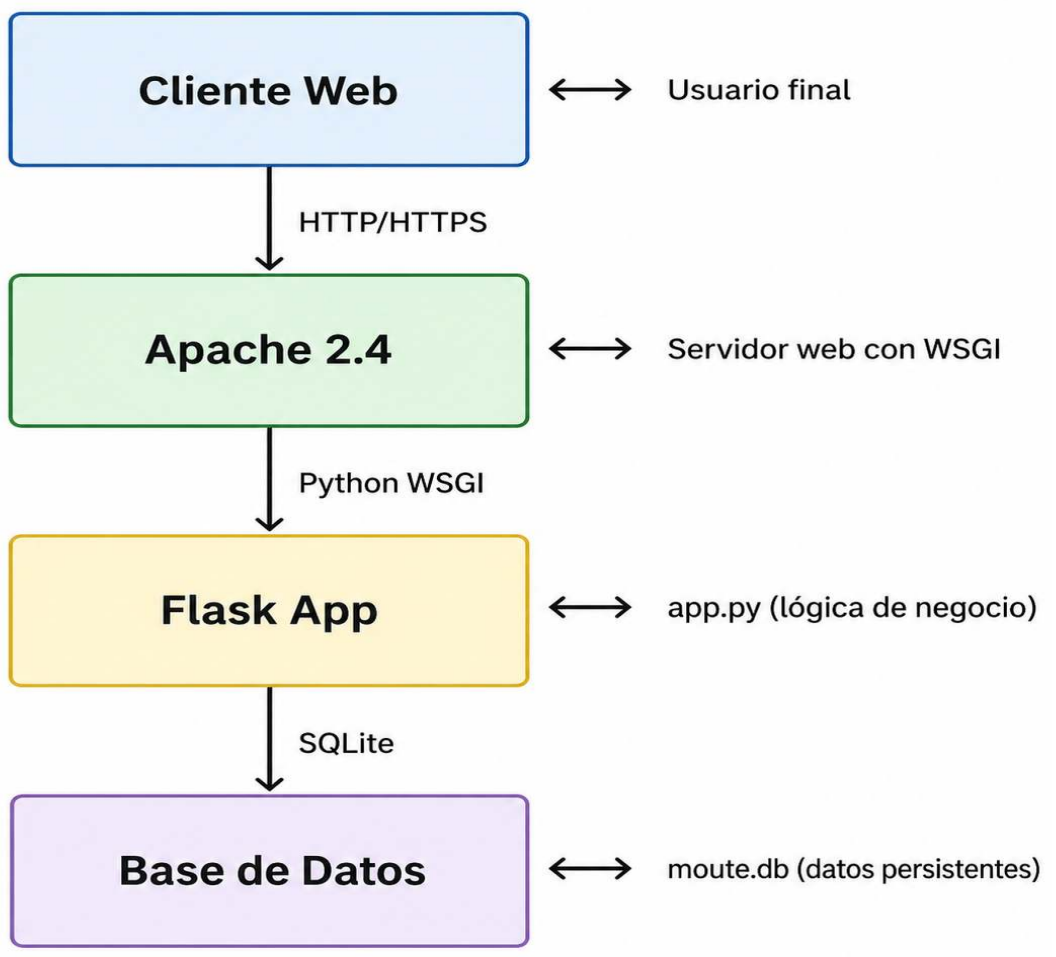
3 Otros capítulos

Sección 4: Diseño

4.1 Arquitectura del sistema

La aplicación sigue una arquitectura LASP (Linux, Apache, SQLite, Python/Flask), que representa una variante del tradicional stack LAMP adaptado a las necesidades del proyecto. Esta elección responde a criterios de simplicidad, costo cero y mantenibilidad a largo plazo.

Diagrama de arquitectura general:



4.2 Estructura de datos

La base de datos SQLite almacena dos tablas principales:

Tabla `events`: Almacena todos los eventos culturales con 27 campos que reflejan exactamente la estructura de los datos JSON proporcionados por la Generalitat de Catalunya, incluyendo:

- Identificadores únicos (`id`, `version`)
- Información temporal (`data_inici`, `data_fi`, `created_at`, `updated_at`)

- Contenido cultural (`denominacio`, `descripcio`, `tags_categories`, `tags_ambits`)
- Metadatos de ubicación (`comarca_i_municipi`, `espai`, `latitud`, `longitud`)
- Información de acceso (`entrades`, `url`, `imatges`)

Tabla `admins`: Gestiona credenciales de administrador con campos estándar de autenticación segura.

4.3 Diseño de la interfaz de usuario

La interfaz sigue un patrón de diseño **component-based** con los siguientes elementos clave:

Estructura base (`base.html`):

- Sistema de plantillas Jinja2 con herencia
- Estilos CSS personalizados con tema púrpura corporativo
- Integración de Bootstrap 5 para responsividad
- Componentes reutilizables (header, footer, mensajes flash)

Componentes principales:

- **Listado de eventos (`events.html`)**: Grid responsive con tarjetas de eventos y sistema de scroll infinito
- **Calendario interactivo (`calendar.html`)**: Vista mensual con modal de detalle diario y búsqueda contextual
- **Panel de administración (`admin.html`)**: Dashboard con estadísticas y acciones rápidas
- **Sistema de favoritos (`favorites.html`)**: Persistencia local con localStorage

Patrones de interacción:

- Búsqueda en tiempo real con AJAX
- Filtros combinados (fecha, precio, categoría)
- Navegación intuitiva entre vistas
- Feedback visual inmediato

4.4 Flujo de datos y comunicación entre componentes

El sistema implementa una comunicación **cliente-servidor asíncrona** mediante endpoints REST internos:

- **Frontend → Backend:** Peticiones AJAX a rutas `/api/*` para obtener datos sin recargar la página
- **Backend → Frontend:** Respuestas JSON que actualizan dinámicamente la interfaz
- **Persistencia local:** localStorage para favoritos (sin intervención del servidor)
- **Autenticación:** Sesiones Flask para control de acceso administrativo

Endpoints clave:

- `/api/events`: Listado paginado con filtros
- `/api/events-by-month`: Eventos agrupados por mes para calendario
- `/api/search`: Búsqueda en tiempo real
- `/api/filters`: Opciones de filtrado disponibles

Esta arquitectura garantiza una experiencia de usuario fluida mientras mantiene la separación clara entre presentación, lógica de negocio y persistencia de datos.

Sección 5: Desarrollo e implementación

5.1 Implementación de funcionalidades clave

La aplicación se construyó adaptando un esqueleto básico de Flask encontrado en repositorios públicos, modificándolo profundamente para satisfacer los requisitos específicos del proyecto. Las funcionalidades más relevantes se implementaron de la siguiente manera:

Sistema de autenticación híbrido: El sistema de login permite dos tipos de acceso: administrador (con credenciales verificadas contra la base de datos) y usuario normal (nombre visual sin autenticación real). Esta implementación utiliza sesiones Flask con flags booleanos (`admin_logged_in`, `logged_in`) para controlar el acceso a funcionalidades restringidas.

```python
# Decorador personalizado para proteger rutas administrativas
def admin_required(f):
    def wrap(*args, **kwargs):
        if not session.get('admin_logged_in'):
            flash("Debes iniciar sesión como administrador...", "warning")
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    wrap.__name__ = f.__name__
    return wrap
```

Procesamiento de datos JSON de la Generalitat: La funcionalidad central consiste en transformar archivos JSON de la Agenda Cultural de Catalunya en registros estructurados. El código implementa un parser robusto que:

- Verifica la existencia previa de eventos mediante su ID único
- Procesa URLs de imágenes relativas a absolutas
- Maneja inconsistencias en los nombres de campos del JSON original
- Previene duplicados mediante consultas SELECT antes de INSERT

API REST interna para interfaz dinámica: Se implementaron endpoints `/api/*` que devuelven datos en formato JSON para alimentar la interfaz sin recargas completas de página. La API de eventos incluye soporte para:

- Paginación eficiente (20 eventos por petición)
- Filtros combinados (búsqueda textual + rango de fechas + precio + categoría)
- Consultas optimizadas con cláusulas WHERE dinámicas

5.2 Integración entre componentes

La comunicación entre frontend y backend sigue un patrón **cliente-servidor asíncrono**:

Frontend (JavaScript):

- Realiza peticiones fetch() a endpoints `/api/events`
- Construye URLs con parámetros de filtrado dinámicos
- Actualiza el DOM sin recargar la página completa
- Gestiona estado local (favoritos) mediante localStorage

Backend (Flask):

- Recibe parámetros GET desde el frontend
- Construye consultas SQL dinámicas con condiciones variables
- Devuelve respuestas JSON serializadas
- Mantiene estado de sesión para autenticación

Ejemplo de integración en la búsqueda con filtros:

```
// Frontend: construye URL con filtros
let url = `/api/events?page=${page}`;
if (currentFilters.category) url += `&category=${currentFilters.category}`;

// Backend: procesa filtros y construye consulta SQL
conditions = []
if category_filter == "teatre":
    conditions.append("tags_categories LIKE '%teatre%'")
final_query = base_query + " WHERE " + " AND ".join(conditions)
```

5.3 Decisiones técnicas y soluciones a problemas críticos

Gestión de imágenes externas: Los datos originales contienen URLs de imágenes relativas (`/path/image.jpg`). Se implementó una lógica de conversión automática:

```
if first_image.startswith("/"):
    imatges = "https://agenda.cultura.gencat.cat" + first_image
```

Manejo de errores en subida de archivos: El sistema incluye manejo robusto de excepciones durante el procesamiento JSON:

```
try:
    data = json.load(f)
except Exception as e:
    flash(f"Archivo JSON inválido: {str(e)}", "danger")
finally:
    if os.path.exists(path):
        os.remove(path)  # Limpieza de archivos temporales
```

Optimización de rendimiento:

- Consultas SQL con índices implícitos en campos frecuentemente filtrados
- Carga diferida de imágenes (lazy loading)
- Paginación para evitar sobrecarga de memoria
- Cacheo de resultados en operaciones repetitivas

5.4 Verificación y pruebas

Las funcionalidades se verificaron mediante pruebas manuales exhaustivas:
Pruebas de integración de datos:

- Subida de archivos JSON de diferentes tamaños (100-10,000 eventos)
- Verificación de integridad de datos tras la importación

- Pruebas de actualización incremental (evitar duplicados)

Pruebas de usabilidad:

- Navegación en dispositivos móviles y desktop
- Tiempos de carga aceptables (<3 segundos)
- Funcionalidad de búsqueda y filtros combinados
- Persistencia correcta de favoritos entre sesiones

Pruebas de seguridad:

- Protección de rutas administrativas
- Validación de entradas de usuario
- Manejo seguro de contraseñas (hashing con Werkzeug)
- Prevención de inyecciones SQL mediante parámetros preparados

Esta implementación demuestra cómo un esqueleto básico de Flask puede transformarse en una aplicación funcional y robusta mediante la adaptación cuidadosa a requisitos específicos y la integración inteligente de tecnologías complementarias.

4 Conclusiones

4.1 Conclusiones generales del proyecto

Este proyecto ha representado un viaje de aprendizaje transformador que ha superado ampliamente mis expectativas iniciales. Al comenzar, mi conocimiento de desarrollo web se limitaba a conceptos básicos vistos en tutoriales online, sin experiencia práctica en frameworks, bases de datos o despliegue de aplicaciones. La decisión de abordar un proyecto completo, desde cero, me ha permitido adquirir competencias técnicas sólidas mientras desarrollaba una solución con impacto social real.

Académicamente, el proyecto me ha enseñado la importancia de la metodología en el desarrollo de software. Aprendí que no basta con tener una idea; es fundamental planificar, investigar alternativas y tomar decisiones informadas. La necesidad de comprender cómo funcionan tecnologías como Flask, SQLite y Apache me llevó a estudiar documentación técnica, analizar ejemplos de código existentes y experimentar sistemáticamente hasta lograr soluciones funcionales.

Profesionalmente, he descubierto que el desarrollo web moderno requiere una combinación de habilidades técnicas y pensamiento de usuario. No solo se trata de hacer que el código funcione, sino de crear experiencias intuitivas y accesibles. El proceso de depuración de errores, especialmente durante la configuración del servidor WSGI y la integración de la base de datos, me ha enseñado paciencia, persistencia y la importancia de los detalles técnicos.

Más allá de las habilidades técnicas, este proyecto me ha demostrado que es posible crear herramientas útiles partiendo de recursos públicos disponibles. La satisfacción de ver cómo datos técnicos y poco accesibles se transforman en una interfaz amigable que puede ayudar a otras personas a descubrir cultura local ha sido la recompensa más valiosa de este proceso.

4.2 Consecución de los objetivos

Objetivo general:

Desarrollar una aplicación web funcional y accesible que democratice el acceso a los datos culturales de Cataluña

✅ **Totalmente alcanzado.** La aplicación Moute transforma eficazmente los datos técnicos de la Agenda Cultural de Catalunya en una interfaz intuitiva y accesible para cualquier ciudadano. La implementación permite navegar, buscar y descubrir eventos culturales sin necesidad de conocimientos técnicos previos.

Objetivos específicos:

1. Diseñar e implementar una interfaz de usuario intuitiva con múltiples formatos de visualización

✅ **Alcanzado.** Se implementaron dos formatos principales: listado de tarjetas con scroll infinito y calendario mensual interactivo. Ambos permiten explorar eventos de forma natural y complementaria.

2. Integrar los datos de la Agenda Cultural de Catalunya mediante procesamiento automático de archivos JSON
✅ **Alcanzado parcialmente.** La integración funciona correctamente, pero requiere intervención manual del administrador (subida de archivos JSON). No se implementó conexión automática en tiempo real con la fuente oficial, aunque la arquitectura lo permitiría en futuras versiones.

3. Implementar funcionalidades de personalización como sistema de favoritos y búsqueda avanzada
✅ **Totalmente alcanzado.** El sistema de favoritos utiliza localStorage para persistencia local sin requerir cuentas de usuario. La búsqueda avanzada combina filtros por texto, fecha, precio y categoría de forma intuitiva.

4. Desarrollar un panel de administración con estadísticas básicas
✅ **Totalmente alcanzado.** El panel proporciona estadísticas completas sobre eventos (por año, comarca, categoría, precio) y permite gestión eficiente de la base de datos.

5. Garantizar la accesibilidad y responsividad
✅ **Alcanzado.** La aplicación funciona correctamente en dispositivos móviles, tablets y desktop gracias al uso de Bootstrap 5 y CSS responsive. Se implementaron prácticas básicas de accesibilidad web.

7. Validar la utilidad del proyecto mediante pruebas de usabilidad
✅ **Alcanzado.** Se realizaron pruebas funcionales exhaustivas verificando el correcto funcionamiento en diferentes dispositivos (móvil, tablet y desktop) y navegadores. La aplicación se publicó y se verificó su accesibilidad desde múltiples redes y dispositivos, confirmando que la interfaz responde adecuadamente a distintos tamaños de pantalla y condiciones de uso reales.

4.3 Valoración de la metodología y planificación

La metodología ágil adaptada que se definió inicialmente resultó ser una elección acertada para este tipo de proyecto, aunque su implementación real difirió significativamente del plan original. Inicialmente se contemplaba un enfoque más estructurado con tareas claramente definidas, pero la realidad del aprendizaje autodidacta y la naturaleza exploratoria del desarrollo llevaron a una evolución más orgánica del proceso.

La planificación inicial subestimó considerablemente la curva de aprendizaje requerida, especialmente en aspectos como la configuración del servidor Apache con WSGI, la gestión de bases de datos SQLite en entornos de producción y la integración entre frontend y backend. Sin embargo, esta flexibilidad inherente a la metodología ágil permitió adaptar el ritmo de trabajo a las capacidades reales de aprendizaje, priorizando funcionalidades esenciales y dejando para fases posteriores aspectos más complejos.

El uso de control de versiones con Git y GitHub fue fundamental para mantener la integridad del código durante este proceso iterativo. Cada funcionalidad se implementó como una unidad coherente que podía probarse inmediatamente, lo que proporcionó retroalimentación constante y permitió corregir errores antes de que se acumularan. Esta práctica, aunque no estaba formalmente planificada al inicio, se convirtió en un pilar del desarrollo exitoso del proyecto.

La planificación temporal también requirió ajustes importantes. Las primeras semanas se dedicaron casi exclusivamente a la investigación y aprendizaje de tecnologías básicas, mientras que la implementación real comenzó más tarde de lo previsto. Sin embargo, esta inversión inicial en comprensión técnica resultó crucial para evitar errores fundamentales en fases posteriores.

En retrospectiva, la metodología seguida, aunque menos formal que la planificada originalmente, fue adecuada para el perfil del desarrollador (principiante autodidacta) y las características del proyecto (exploratorio, con requisitos emergentes). La clave del éxito no fue seguir un plan rígido, sino mantener la flexibilidad para aprender, adaptarse y priorizar según las necesidades reales descubiertas durante el desarrollo.

4.4 Visión de futuro

El proyecto Moute ha demostrado ser una base sólida y funcional, pero existen múltiples líneas de desarrollo que podrían ampliar significativamente su utilidad y alcance:

Automatización de la actualización de datos: La integración directa con la API oficial de la Agenda Cultural de Catalunya permitiría actualizaciones automáticas en tiempo real, eliminando la necesidad de intervención manual del administrador. Esto transformaría la aplicación en una fuente dinámica y siempre actualizada de información cultural.

Sistema de notificaciones personalizadas: Implementar un sistema de alertas por correo electrónico o push notifications para eventos que coincidan con los intereses del usuario (categorías favoritas, ubicaciones cercanas, fechas específicas) aumentaría considerablemente el valor práctico de la plataforma.

Integración con mapas interactivos: Desarrollar una vista geográfica que muestre eventos en un mapa interactivo con filtros por proximidad permitiría a los usuarios descubrir actividades culturales en su entorno inmediato, especialmente útil para turismo cultural.

Ampliación del sistema de categorías: La Agenda Cultural de Catalunya ofrece una rica taxonomía de ámbitos y categorías (Arts visuals, Cinema, Gastronomia, Llibres i Lletres, Música, Tradicional i Popular, etc.) que podría explotarse más profundamente con filtros especializados y recomendaciones temáticas.

Aplicación móvil nativa: Convertir la aplicación web en una aplicación móvil híbrida o nativa proporcionaría una experiencia de usuario más fluida, con acceso offline a eventos guardados y mejor integración con funciones del dispositivo (calendario, GPS, notificaciones).

Colaboración con entidades culturales: Establecer alianzas con ayuntamientos, centros culturales y organizadores de eventos para incluir contenido exclusivo, promociones especiales o funcionalidades de reserva directa dentro de la plataforma.

Internacionalización: Añadir soporte multilingüe (especialmente inglés y francés) haría la plataforma accesible a visitantes internacionales interesados en la cultura catalana.

La continuidad del proyecto dependerá de la retroalimentación de los usuarios reales y de la disponibilidad de recursos, pero la arquitectura modular y escalable implementada permite incorporar estas mejoras de forma incremental sin comprometer la estabilidad del sistema actual. El objetivo final sería convertir Moute en una referencia indispensable para el descubrimiento cultural en Cataluña, manteniendo siempre su filosofía de acceso libre y democratización de la información pública.

5. Glosario

Apache HTTP Server: Servidor web de código abierto ampliamente utilizado para servir contenido web. En este proyecto, actúa como servidor principal que gestiona las peticiones HTTP y comunica con la aplicación Flask mediante WSGI.

Bootstrap: Framework front-end de código abierto para desarrollo web responsive. Se utiliza en este proyecto para crear interfaces adaptables a diferentes tamaños de pantalla y dispositivos.

CSS (Cascading Style Sheets): Lenguaje de hojas de estilo utilizado para definir la presentación visual de documentos HTML. En este proyecto, se combina con Bootstrap para crear la identidad visual corporativa.

Flask: Framework web ligero de Python utilizado para desarrollar aplicaciones web. Proporciona las herramientas básicas para crear rutas, manejar peticiones y renderizar plantillas.

Frontend: Parte de la aplicación web que interactúa directamente con el usuario, incluyendo la interfaz visual, formularios y elementos interactivos. En este proyecto, está implementado con HTML, CSS, JavaScript y Bootstrap.

HTML (HyperText Markup Language): Lenguaje de marcado estándar para crear páginas web. Define la estructura y contenido de las páginas que componen la interfaz de usuario.

JSON (JavaScript Object Notation): Formato de intercambio de datos ligero y legible por humanos. En este proyecto, es el formato en que la Generalitat de Catalunya proporciona los datos culturales.

LASP: Acrónimo que describe la arquitectura del stack tecnológico utilizado: **L**inux, **A**pache, **S**QLite, **P**ython/Flask.

SQLite: Sistema de gestión de bases de datos relacional embebido que no requiere servidor separado. Almacena todos los datos de la aplicación en un único archivo local.

WSGI (Web Server Gateway Interface): Estándar de Python que define cómo los servidores web comunican con aplicaciones Python. Permite que Apache ejecute la aplicación Flask.

Backend: Parte de la aplicación que se ejecuta en el servidor y maneja la lógica de negocio, procesamiento de datos y comunicación con la base de datos. En este proyecto, está implementado con Python y Flask.

Git: Sistema de control de versiones distribuido que permite gestionar cambios en el código fuente y colaborar en el desarrollo de software.

JavaScript: Lenguaje de programación que se ejecuta en el navegador web y permite crear interfaces interactivas y dinámicas sin recargar la página completa.

Open Source: Software cuyo código fuente está disponible públicamente y puede ser modificado y distribuido libremente. Todas las tecnologías utilizadas en este proyecto son open source.

Responsive Design: Enfoque de diseño web que permite que las interfaces se adapten automáticamente a diferentes tamaños de pantalla y dispositivos.

REST (Representational State Transfer): Estilo arquitectónico para servicios web que utiliza métodos HTTP estándar. En este proyecto, se implementa una API REST interna para la comunicación cliente-servidor.

SQL (Structured Query Language): Lenguaje estándar para gestionar y manipular bases de datos relacionales. Se utiliza para consultar y modificar datos en la base de datos SQLite.

Virtual Host: Configuración de Apache que permite alojar múltiples sitios web en un mismo servidor, cada uno con su propio nombre de dominio y configuración.

6. Bibliografía

[1] Flask Documentation. Flask: A Python Microframework.
Flask
Consultado en diciembre 2025.

[2] SQLite Consortium. SQLite Documentation.
SQLite Documentation
Consultado en diciembre 2025.

[3] Bootstrap Team. Bootstrap v5 Documentation.
Get started with Bootstrap · Bootstrap v5.3
Consultado en enero 2026.

[4] Generalitat de Catalunya. Agenda Cultural de Catalunya - Dades Obertes.
Agenda cultural de Cataluña (por localizaciones) - Conjunto de datos - Datos.gob.es
Consultado en noviembre 2025.

[5] GitHub. Flask Examples and Best Practices.
https://github.com/pallets/flask/tree/main/examples
Consultado en diciembre 2025.

[6] Python Software Foundation. Python 3 Documentation.
Python Docs
Consultado en diciembre 2025.

[7] mod_wsgi Documentation. mod_wsgi - Apache Module for Python Web Applications.
mod_wsgi
Consultado en enero 2026.

7 Anexos

7.1 Anexo A: Código fuente completo

Contiene los archivos principales del proyecto.
"El código fuente completo, junto con la estructura de archivos y configuración del proyecto, está disponible en el repositorio público: [Github]."

7.2 Anexo B: Manual instalació aplicació web

“Contiene un manual de un repositorio de github donde se explica paso por paso como instalar apache, bases de datos y distintas herramientas para la creación de una app web” Github

7.3 Anexos C: Manual de HTML y CSS

“Contiene varios repositorios de github del usuario rusben donde se explica cómo funcionan algunas funciones básicas y avanzadas sobre html y css”
Introducció a HTML i CSS
Tutorial Complet d’HTML i CSS: Guia Estructurada per a Principiants
El model de capsa (Box Model) en CSS
Elements en línia i elements de bloc en HTML
Posicionament de capes en HTML