Rave Daddy

Desarrollo de Aplicaciones Web

|  |  |
| --- | --- |
| Trabajo fin de ciclo presentado por: | Jose Mª Rubio Robles |
| Tutor/a: | Ramon Merchán Sanzano |
| Fecha: | 13 diciembre de 2025 |

**IES Juan de la Cierva**

Departamento de Informática

Memoria del proyecto

Documento técnico

Índice

[1. Introducción 3](#_Toc216199806)

[1.1. Descripción y contexto del proyecto 3](#_Toc216199807)

[1.2. Motivación del proyecto 3](#_Toc216199808)

[1.3. Beneficios esperados 3](#_Toc216199809)

[2. Objetivos 4](#_Toc216199810)

[2.1. Objetivos generales 4](#_Toc216199811)

[2.2. Objetivos específicos 5](#_Toc216199812)

[3. Contexto actual 5](#_Toc216199813)

[3.1. Estado del arte. 5](#_Toc216199814)

[3.2. Conceptos clave. 6](#_Toc216199815)

[4. Planificación 7](#_Toc216199816)

[4.1. Metodología de gestión del proyecto 7](#_Toc216199817)

[4.2. Acciones y tareas 8](#_Toc216199818)

[4.3. Temporización y secuenciación 9](#_Toc216199819)

[5. Análisis de requisitos 10](#_Toc216199820)

[5.1. Diagrama de casos de uso. 11](#_Toc216199821)

[5.2. Requisitos funcionales/negocio 11](#_Toc216199822)

[5.3. Requisitos no funcionales. 12](#_Toc216199823)

[5.4. Descripción de los usuarios y sus necesidades. 13](#_Toc216199824)

[6. Diseño de la aplicación 13](#_Toc216199825)

[6.1. Mockups o wireframes o prototipos de la interfaz gráfica de usuario. 13](#_Toc216199826)

[6.2. Arquitectura del sistema. 15](#_Toc216199827)

[6.3. Diagramas Entidad/Relación. 16](#_Toc216199828)

[6.4. Diagrama de Clases. 17](#_Toc216199829)

[7. Desarrollo de la aplicación 18](#_Toc216199830)

[7.1. Tecnologías y herramientas utilizadas 18](#_Toc216199831)

[7.2. Descripción de las principales funcionalidades 19](#_Toc216199832)

[8. Pruebas y Validación. 21](#_Toc216199833)

[8.1. Pruebas unitarias 21](#_Toc216199834)

[8.2. Pruebas de integración. 23](#_Toc216199835)

[8.3 Pruebas funcionales 23](#_Toc216199836)

[9. Despliegue y mantenimiento. 24](#_Toc216199837)

[9.1. Despliegue de la aplicación 24](#_Toc216199838)

[9.2 Mantenimiento. 25](#_Toc216199839)

[10. Conclusiones y trabajo futuro. 25](#_Toc216199840)

[10.1. Trabajo futuro 25](#_Toc216199841)

[11. Relación del proyecto con los módulos del ciclo 26](#_Toc216199842)

[12. Bibliografía/Webgrafía 26](#_Toc216199843)

[13. Anexos 27](#_Toc216199844)

[A. Estructura del proyecto. 27](#_Toc216199845)

[B. Ficheros de configuración Docker. 28](#_Toc216199846)

[Ilustración 1Diagrama de Gantt 10](#_Toc216199847)

[Ilustración 2 Diagrama de casos de uso 12](#_Toc216199848)

[Ilustración 3 Mockup 15](#_Toc216199849)

[Ilustración 4 Home final 15](#_Toc216199850)

[Ilustración 5 Login/Register Modal 15](#_Toc216199851)

[Ilustración 6 Ejemplo salida de eventos 16](#_Toc216199852)

[Ilustración 7 Diagrama de flujo de la aplicación 17](#_Toc216199853)

[Ilustración 8 Diagrama de clases 18](#_Toc216199854)

[Ilustración 9 Llamada al agente con el historial de la conversación 20](#_Toc216199855)

[Ilustración 10 Manejo de historial de la conversación por sesiones. 21](#_Toc216199856)

[Ilustración 11 Webscraping de detalles de eventos 21](#_Toc216199857)

[Ilustración 12 Llamada a base de datos para events\_evento\_detalle 22](#_Toc216199858)

[Ilustración 13 Salida de base de datos de la Ilustración 12 22](#_Toc216199859)

[Ilustración 14 Salida del cron para obtención de eventos 23](#_Toc216199860)

[Ilustración 15 Función de webscraping de detalles del evento 23](#_Toc216199861)

[Ilustración 16 Función que guarda los detalles del evento 23](#_Toc216199862)

[Ilustración 17 Respuesta del agente a la pregunta sobre el evento "ZFX" 24](#_Toc216199863)

[Ilustración 18 Salida de base de datos al buscar el evento ZFX 24](#_Toc216199864)

[Ilustración 19 Salida al buscar el detalle del evento con id 51 24](#_Toc216199865)

[9 Ilustración 20 Mensaje que aparece cuando te identificas con tu cuenta 24](#_Toc216199866)

[Ilustración 21 Estructura del proyecto 28](#_Toc216199867)

[Ilustración 22 Dockerfile 29](#_Toc216199868)

[Ilustración 23 Contenedor de la base de datos 29](#_Toc216199869)

[Ilustración 24 Contenedor web 30](#_Toc216199870)

# Introducción



# Descripción y contexto del proyecto

Ravedaddy es una aplicación de web desarrollada con la finalidad de proporcionar a los usuarios un espacio consolidado en donde accedan a la información sobre los eventos y fiestas de música electrónica que se organizan en su entorno. Nace para solucionar la percepción muy común de la sensación de "perderse algo" en el momento de salir en la escena de la música electrónica, ofreciendo un instrumento que reporta información actualizada. También, tiene la intención de ser un punto de partida para aquellas personas que pretenden introducirse en el universo de la música electrónica, proporcionando una experiencia asequible y estructurada.

# Motivación del proyecto

La motivación principal de este proyecto viene de mi afición a la música, específicamente, la música electrónica. Además de mi afición por la música, nace de mi curiosidad por el funcionamiento de la inteligencia artificial. La combinación de estos dos intereses da lugar a Rave Daddy. Un proyecto que me permite explorar cómo integrar la IA dentro de un proyecto un estrecha relación a la música.

# Beneficios esperados

Los principales beneficios esperados son el aprender a estructurar y planificar un proyecto completo, intentando llegar en forma exacta al cumplimiento de los objetivos. Además, se busca comprender el funcionamiento práctico de un agente de IA a pequeña escala e intentar entender como interactúa con los diferentes elementos que se le pongan a su disposicion. En conjunto, el proyecto pretende aportar tanto una mejora técnica a nivel de planificación como una mejor comprensión del proceso de desarrollo web con IA.

# Objetivos

El proyecto tiene como objetivo final desarrollar una aplicación web capaz de centralizar información sobre eventos y ofrecer un asistente basado en inteligencia artificial que interactúe con el usuario de forma natural. Para conseguirlo, se establecen una seria de objetivos generales y específicos que guiaran el desarrollo técnico del sistema.

# Objetivos generales

* Desarrollar un sistema de obtención y actualización automática de eventos y DJs usando técnicas de webscraping y ejecución programada de tareas por cron.
* Implementar un agente inteligente capaz de acceder a la base de datos, consultar información y ejecutar funciones de webscraping garantizando una interacción correcta con el usuario.
* Integrar un modelo de lenguaje (LLM) basado en Gemini para conseguir respuestas comprensibles para el usuario.
* Desarrollar un sistema de sesiones para el agente utilizando las sesiones dentro de las cookies.
* Diseñar una aplicación web funcional, sencilla e intuitiva que permita al usuario consultar eventos con rapidez.
* Utilizar como base la plataforma de Docker para empaquetar la aplicación y hacerla lo más portable posible.



# Objetivos específicos

* Analizar e identificar los requisitos de tipo funcional y no funcional de la aplicación para orientar su desarrollo.
* Diseñar los modelos de datos que se requieren para almacenar información de eventos, DJs y consultas.
* Implementar un webscraping de Xceed de forma que se obtengan eventos, detalles y DJs y se estructure correctamente la información obtenida.
* Utilizar Playwright para el webscraping.
* Configurar un cron de forma que se automatice la obtención periódica de los datos obtenidos.
* Diseñar una interfaz web que sea clara y sencilla, teniendo como objetivo una experiencia de usuario clara y directa.
* Integrar el LLM Gemini con LangChain desarrollando un prompt de sistema y un agente que pueda seleccionar la herramienta adecuada para cada ejecución.
* Gestionar la memoria conversacional mediante sesiones para permitir la continuidad de la interacción con el usuario.
* Implementar el acceso a la consulta de temperatura desde API externa para ampliar la información proporcionada sobre cada evento.
* Realizar pruebas funcionales y de integración verificando el correcto funcionamiento del webscraping, del agente IA y de las APIs desarrolladas.

# Contexto actual

# Estado del arte.

Rave Daddy se sitúa en un punto intermedio entre dos tipos de soluciones ampliamente extendidas: los chatbots conversacionales y las aplicaciones de búsqueda y reserva de eventos. En el ámbito de los chatbots, existen múltiples referentes consolidados como ChatGPT o distintos copilots disponibles en el mercado, que destacan por su capacidad de interacción natural y por el uso de modelos de lenguaje avanzados. Aunque estas herramientas poseen un alto nivel de computación, no están orientadas de forma específica a la gestión de eventos musicales ni integran funciones propias de plataformas de booking.

Por otra parte, aplicaciones que giran en torno a una búsqueda y una reserva de experiencias, como Booking y similares, constituyen el modelo más análogo en el que se mezclan la consulta de información y el filtrado de contenidos. No obstante, la mayoría de estos mismos tipos de plataformas sólo ofrecen chatbots conversacionales muy restringidos, pero no con la adecuada integración con agentes que permitan ejecutar herramientas, procesar información o personalizar la búsqueda de eventos.

Hasta donde ha llegado la revisión realizada no se han podido encontrar propuestas que hagan uso de una manera u otra de forma conjunta, un sistema conversacional basado en IA y una plataforma de descubrimiento de eventos de música electrónica. La inexistencia de soluciones híbridas permite que Rave Daddy sobresalga en el sector. No es un océano azul, pero la combinación de un chatbot funcional y una aplicación de búsqueda de eventos cubren un terreno poco explorado. Además, el mercado de eventos electrónicos es un mercado muy fragmentado, puesto que la información está dispersa entre clubes, promotores y páginas independientes, lo que hace que el proyecto tenga cabida y oportunidad dentro el mercado.

# Conceptos clave.

* Inteligencia artificial generativa: Tecnología basada en modelos de lenguaje capaces de comprender y generar texto, utilizada para permitir la interacción conversacional entre el usuario y el sistema.
* Agente IA: Componente inteligente que emplea herramientas para obtener información y responder de forma contextualizada según las instrucciones del LLM.
* LLM (Large Language Model o Modelo de lenguaje de gran escala): En este caso Gemini 2.5 flash-lite. Procesa las entradas del usuario y decide que acciones debe ejecutar el agente para generar una respuesta con sentido.
* Webscraping: Técnica automatizada para extraer datos de páginas web, utilizada para conseguir información sobre los eventos y los DJs de plataformas externas.
* API REST: Interfaz de programación de aplicaciones que sigue los principios de la arquitectura de Transferencia de Estado Representacional (REST) para conectar sistemas de manera flexible y estandarizada, principalmente a través de solicitudes y respuestas HTTP.
* Sistemas de programación de tareas (Cron): herramienta en segundo plano utilizado en sistemas tipo Unix para programar y automatizar la ejecución de tareas repetitivas en momentos o intervalos específicos mediante el uso de archivos crontab.
* Base de datos relacional: es un sistema que organiza y almacena datos estructurados que se interconectan mediante relaciones predefinidas e identificadores únicos. Guardando información persistente.
* Framework: es una estructura predefinida con herramientas, librerías y convenciones que sirve de base para desarrollar software. En este caso el framework de Python, Django
* Docker: Plataforma basada en contenedores e imágenes que permite ejecutar proyectos en entornos aislados, facilitando el despliegue y la portabilidad de este.

# Planificación

# Metodología de gestión del proyecto

El desarrollo del proyecto se ha gestionado siguiendo una metodología basada en SCRUM, adaptada al contexto de un proyecto individual. Cada quince días se han convocado reuniones con el tutor, que han servido como puntos de control para valorar el progreso, detectar incidencias y redefinir prioridades.

Aunque no ha existido un equipo de trabajo, se ha mantenido una organización estructurada del proyecto mediante la división en tareas y objetivos parciales, simulando ciclos de trabajo a sprints como dice el método SCRUM.

El control de versiones se ha conseguido mediante el software git y utilizando la nube de GitHub, consiguiendo un repositorio con un histórico de cambios que ha permitido mantener el control del estado del proyecto, gestionar versiones estables y volver a estados anteriores en caso de necesidad.

La gestión de incidencias y el seguimiento del progreso se ha realizado de forma continua, ajustando los tiempos y prioridades conforme avanzaba el desarrollo.

# Acciones y tareas

El desarrollo del proyecto se ha llevado a cabo a lo largo de aproximadamente dos meses y medio. Durante las primeras semanas, el tiempo dedicado fue de entre ocho y diez horas semanales, mientras que en el último mes el tiempo llego hasta aproximadamente veinte horas semanales y con algún fin de semana más, debido a la carga de trabajo asociada a la implementación de las funcionalidades principales, las pruebas y la documentación.

Primero, se realizó una fase de toma de requisitos en la que se definieron los objetivos generales del sistema, el alcance del proyecto y las necesidades principales que debía cubrir la aplicación. Esta fase fue muy importante ya que la finalidad fue crear una base clara para orientar las decisiones técnicas que vayan surgiendo. Posteriormente, se desarrolló la fase de análisis de requisitos, en la que se estudiaron en detalle los aspectos funcionales y no funcionales de la aplicación. Durante esta etapa se revisaron distintas alternativas técnicas y se evaluaron las tecnologías más adecuadas para su implementación. Una vez definidos los requisitos, se llevó a cabo la configuración del entorno de desarrollo y la preparación de la infraestructura mediante contenedores Docker, lo que permitió garantizar la portabilidad y homogeneidad del entorno de ejecución. La fase de diseño incluyó la definición de la arquitectura general del sistema, el diseño de los modelos de datos y la planificación de los principales flujos de interacción entre el usuario y la aplicación.

Terminada toda la parte técnica, inicie los trabajos de codificación, desarrollando la estructura base de la aplicación web y desarrollando los principales módulos, las vistas, los controladores y la lógica de negocio correspondiente. Constatadas las bases, la construcción de la base de la aplicación, se realizó la integración del agente de inteligencia artificial, con lo que se enlazó el sistema con el modelo de lenguaje y se definieron sus tools que son con las que haría peticiones a la base de datos y a diferentes funciones de webscraping para sacar la información deseada. De forma casi paralela se llevó a cabo la implementación del sistema de webscraping encargado de recoger información de la plataforma externa ‘Xceed’ y la programación de tareas automáticas mediante cron para la actualización periódica de eventos por el propio scraping. Las pruebas se ejecutaron de forma continua a lo largo del desarrollo, incluyendo pruebas funcionales, de integración y de validación de resultados, con la finalidad de detectar y corregir errores de forma anticipada. No menos importante, la documentación del proyecto fue construida de forma progresiva desde las fases iniciales hasta su finalización, completándose a medida que se cerraban las distintas etapas del desarrollo, lo cual me ha permitido recoger de forma fiel la evolución real del proyecto.

# Temporización y secuenciación



Ilustración 1Diagrama de Gantt

# Análisis de requisitos

Los requisitos del sistema se dividen en dos grupos, requisitos funcionales y los no funcionales, diferenciando entre lo que el sistema debe hacer y las condiciones técnicas por las que debe funcionar.

**Requisitos funcionales:**

* El sistema debe permitir al usuario consultar directamente al agente por medio de un filtro. Respondiendo el agente con información básica de los eventos
* El sistema debe tener un asistente conversacional con IA con el que el usuario pueda interactuar con naturalidad. Este agente debe poder consultar eventos y buscar artistas por webscraping.
* La aplicación debe realizar la actualización automática de eventos mediante la programación de un cron sin necesidad de la intervención de nadie.
* La aplicación ofrece un sistema opcional de registro e inicio de sesión.

**Requisitos no funcionales:**

* La aplicación se implementa como plataforma web accesible a través de un navegador.
* El despliegue del sistema se realiza mediante contenedores Docker permitiendo la portabilidad de la aplicación.
* El sistema utiliza una base de datos relacional PostgreSQL para la persistencia de datos.
* La aplicación esta orientada principalmente al uso en entornos de escritorio, aunque sea responsive todo el diseño.

# Diagrama de casos de uso.

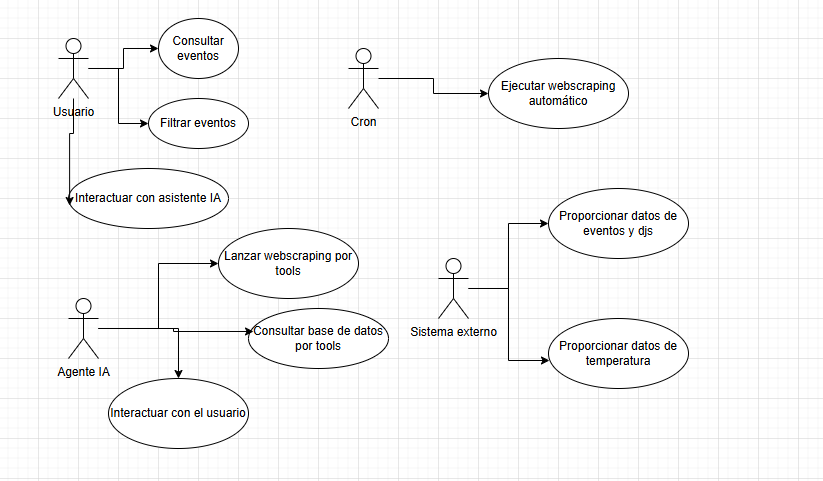


Ilustración 2 Diagrama de casos de uso

# Requisitos funcionales/negocio

La aplicación tiene tres módulos principales que cubren las necesidades del usuario de forma sencilla y directa.  
**Módulo de búsqueda de eventos:**

* Permitir consultar el listado de eventos disponibles.
* Permitir filtrar eventos por nombre.
* Permitir filtrar eventos por fecha.
* Mostrar información básica de cada evento.

**Módulo de asistente de inteligencia artificial:**

* Permitir interactuar con un asistente conversacional con lenguaje natural.
* Permitir al asistente consultar la base de datos de eventos.
* Permitir al asistente ejecutar herramientas internas para obtener información adicional.

**Módulo de autenticación:**

* Permitir el registro opcional de usuarios.
* Permitir el inicio de sesión de usuarios registrados.

# Requisitos no funcionales.

**Requisitos técnicos.**

* La aplicación debe estar implementada como una plataforma web accesible desde un navegador.
* El sistema debe estar desarrollado utilizando el framework Django (Python).
* El sistema debe utilizar una base de datos relacional PostgreSQL para el almacenamiento de datos.
* El despliegue de la aplicación debe realizarse mediante contenedores Docker.

**Requisitos de rendimiento**

El sistema debe ofrecer tiempos de respuesta adecuados para una experiencia fluida del usuario en las operaciones de consulta y filtrado de eventos. Por ello se decidió parar de que el propio agente busque los eventos y sus detalles a una precarga en la base de datos.

**Requisitos de seguridad**

* El sistema debe permitir el acceso tanto a usuarios anónimos como a usuarios registrados.
* El sistema implementa mecanismos básicos de seguridad para evitar accesos no autorizados a las funciones de administración (dadas por Django).
* El sistema debe gestionar las sesiones de usuario de forma segura.

**Requisitos de usabilidad**

* La interfaz debe ser simple, clara e intuitiva.
* El sistema debe priorizar la facilidad de uso frente a la complejidad funcional.

# Descripción de los usuarios y sus necesidades.

En la aplicación existen dos tipos de usuarios, aunque no mantienen mayores diferencias.

**Usuario anónimo**

Es el perfil principal de la plataforma. Puede acceder a la aplicación sin necesidad de registro y consultar los eventos disponibles, así como interactuar con el asistente de inteligencia artificial. Su necesidad principal es acceder de forma rápida y sencilla a información actualizada sobre eventos. Mantiene la información de la conversación por sesión en las cookies.

**Usuario registrado**  
Dispone de las mismas funcionalidades que el usuario anónimo, pero con la posibilidad adicional de crear una cuenta. Este tipo de usuario no requiere funcionalidades adicionales, ya que el acceso a los servicios principales es equivalente al del usuario anónimo.

# Diseño de la aplicación

# Mockups o wireframes o prototipos de la interfaz gráfica de usuario.

La aplicación está diseñada siguiente un enfoque minimalista y centrándose en la claridad para la experiencia conversacional, tomando como referencia el diseño de las aplicaciones web más conocidas de asistentes conversacionales. La aplicación se basa en una única pantalla desde la que el usuario puede interactuar con el agente de IA y filtrar los eventos.

La interfaz esta implementada por un template HTML, renderizada por Django, con su respectivo CSS para mantener una presentación clara y un JS para pequeñas adicciones. Lo que normalmente ocupa otra interfaz como es el login/register esta integrado en el mismo HTML principal mediante un modal emergente que permite mantener al usuario en la misma pantalla todo el rato.  
 El diseño se desarrolló inicialmente mediante wireframes de baja fidelidad con ayuda de generadores IA. Permitiendo definir la estructura básica de la interfaz antes de la creación.

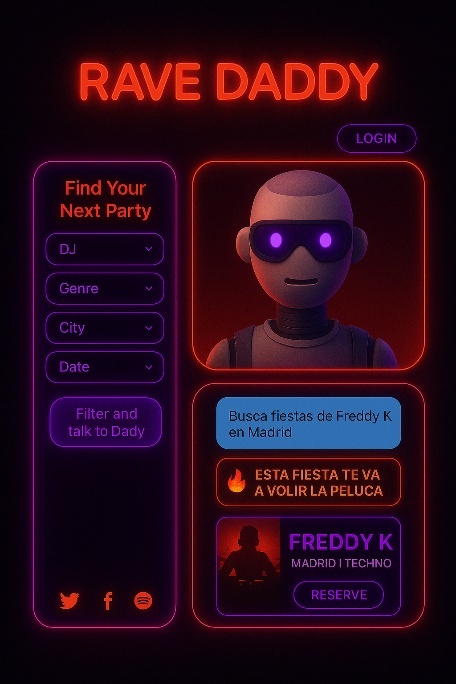


Ilustración 3 Mockup

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Ilustración 4 Home final

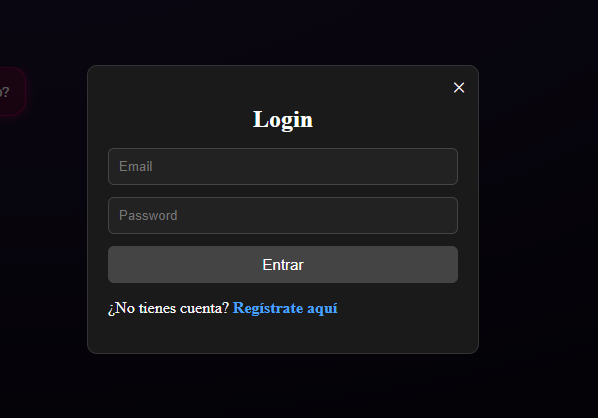


Ilustración 5 Login/Register Modal

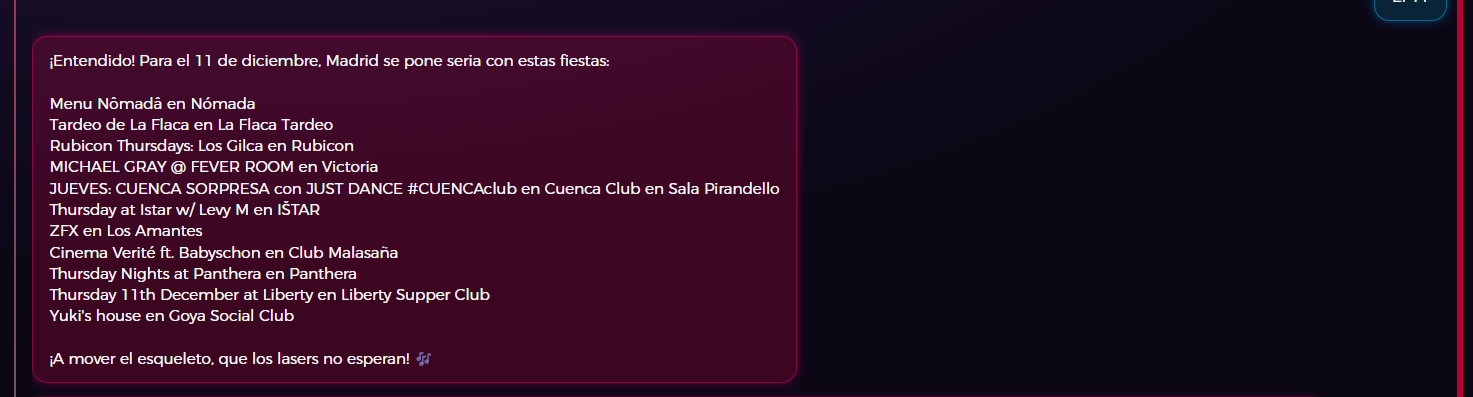


Ilustración 6 Ejemplo salida de eventos

# Arquitectura del sistema.

La aplicación sigue una arquitectura basada Enel Modelo-Vista-Controlador adaptado a el funcionamiento de Django Model-Template-View. Siendo la vista el template y view el controlador.

La arquitectura se compone de los siguientes elementos:

* **Frontend:** plantillas HTML y CSS junto JS renderizado por Django.
* **Backend:** framework Django.
* **Webscraping:** implementado mediante Playwright con la que obtenemos los eventos.
* **Agente IA:** integración con el modelo de LLM Gemini 2.5 flash-lite mediante LangChain.
* **APIs externas:** Como el LLM o la api meteorológica.
* **Automatización:** tareas programadas mediante cron.
* **Despliegue:** contenedores Docker.

**Diagrama

El contenido generado por IA puede ser incorrecto.**

Ilustración 7 Diagrama de flujo de la aplicación

# Diagramas Entidad/Relación.

Diagrama

El contenido generado por IA puede ser incorrecto.

# Diagrama de Clases.

Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.

Ilustración 8 Diagrama de clases

# Desarrollo de la aplicación

# Tecnologías y herramientas utilizadas

La aplicación ha sido desarrollada utilizando una combinación de tecnologías modernas además de la integración de la inteligencia artificial.

**Lenguaje de programación:**

* Python 3.11+ versión recomendada para compatibilidad con Django y librerías de IA

**Framework:**

* Django

**Base de datos:**

* PostgreSQL tipo de base de datos que mejor relación tiene con Django
* Con una librería de conexión psycopg2-binary

**APIs y librerías auxiliares:**

* ‘djangorestframework’ para construir endpoints de tipo REST y conseguir la conexión con el frontend.
* python-dotenv para la gestión de variables de entorno.
* requests para la comunicación con APIs externas.
* langchain==1.0.0 para mantener el proyecto en lo básico y langchain-google-genai para la integración con el modelo de lenguaje Gemini y el manejo del agente.
* langgraph-prebuilt==1.0.4 (versión compatible con langchain) para la representación y ejecución de flujos de herramientas del agente.
* Playwright para hacer webscraping automatizado en páginas externas.
* django-crontab para programar tareas periódicas de actualización de datos.

**Entorno de despliegue (Docker):**

* El propio Docker para asegurar portabilidad y reproducibilidad del entorno.
* En la propia construcción del contenedor se han incluido paquetes esenciales del sistema operativo (libpq-dev, por ejemplo) para garantizar la correcta ejecución del framework Django, Playwright, etc.
* Se incluye la instalación de cron en el contenedor.

**Otras herramientas:**

* Control de versiones mediante Git y por tanto GitHub.
* Chromium como parte de Playwright siendo un navegador comprimido.

# Descripción de las principales funcionalidades

Ilustración 9 Llamada al agente con el historial de la conversación

**Texto

El contenido generado por IA puede ser incorrecto.**

Ilustración 10 Manejo de historial de la conversación por sesiones.

**Texto

El contenido generado por IA puede ser incorrecto.**

Ilustración 11 Webscraping de detalles de eventos

**Texto

El contenido generado por IA puede ser incorrecto.**

# Pruebas y Validación.

# Pruebas unitarias

Ilustración 12 Llamada a base de datos para events\_evento\_detalle



Ilustración 13 Salida de base de datos de la Ilustración 12

Texto

El contenido generado por IA puede ser incorrecto.Aquí se puede comprobar que en la automatización de entradas en la base de datos después de encontrar eventos sale de la manera esperada.

Ilustración 14 Salida del cron para obtención de eventos

Texto

El contenido generado por IA puede ser incorrecto.

Ilustración 15 Función de webscraping de detalles del evento

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Ilustración 16 Función que guarda los detalles del evento

Texto

El contenido generado por IA puede ser incorrecto.

En estas tres partes del código podemos ver como al correr el cron o se crean o se obtienen los detalles de los eventos. Comprobando que funciona el webscraping. Junto con la prueba anterior podemos comprobar que Busca y guarda de forma correcta en la base de datos.

# Pruebas de integración.

Ilustración 17 Respuesta del agente a la pregunta sobre el evento "ZFX"

Patrón de fondo

El contenido generado por IA puede ser incorrecto.

Ilustración 18 Salida de base de datos al buscar el evento ZFX

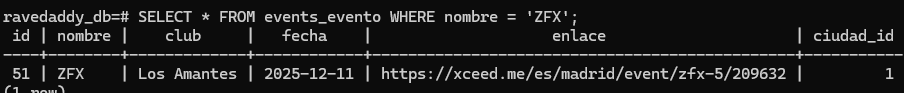
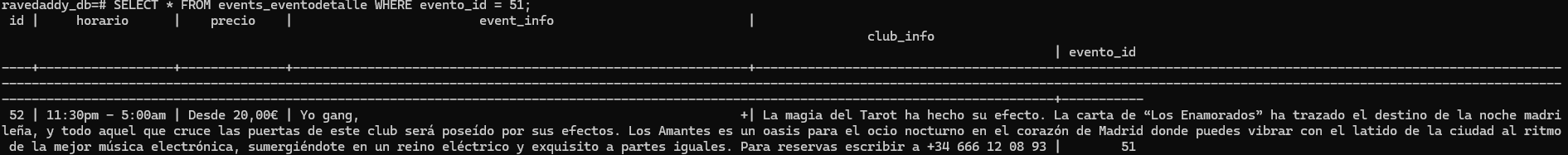


Ilustración 19 Salida al buscar el detalle del evento con id 51



Con estas tres partes del código podemos comprobar que el agente coge de forma correcta la información del evento.

# Pruebas funcionales

1. Ilustración 20 Mensaje que aparece cuando te identificas con tu cuenta



# Despliegue y mantenimiento.

# Despliegue de la aplicación

El despliegue de la aplicación se ha diseñado para realizarse mediante contenedores Docker, como hemos dicho anteriormente, apuntando a conseguir un entorno reproducible, portable y sencillo de desplegar. No se ha llegado a desplegar en un servidor externo. Solo se ha llegado a desplegar mediante Docker por local/LAN.

El proceso de despliegue se basa en los siguientes elementos:

* Un archivo Dockerfile encargado de definir la imagen del contenedor principal.
* El archivo docker-compose.yml que define los componentes y servicios de la aplicación web y base de datos al igual que sus respectivos volúmenes.
* Un script de inicialización entrypoint.sh (lenguaje bash) que gestiona el arranque del servidor, practicando las migraciones e inicializaciones necesarias.

Quedando un flujo final de aplicación:

1. ‘Docker-compose up –build’ llama al docker-compose.yml y crea las imágenes, primero la de la base de datos para que no de ningún error.
2. Se construye la imagen web usando el Dockerfile que también instala las dependencias, además de copiar el archivo de requerimientos para instalar todo lo necesario.
3. Se copia el archivo de inicialización y se lanza en el contenedor web.
4. Este archivo espera a que la base de datos este activa, lanza las migraciones de Django y crea un usuario administrador por defecto.
5. Por último, recoge los estáticos (HTML, CSS y JS), activa el demonio cron además de lanzarlo para coger los primeros datos y levanta el servidor.

Forma de inicializarlo para el administrador:

1. Iniciar Docker
2. En la carpeta Ravedaddy hacer ‘docker-compose up –build’

# Mantenimiento.

El código fuente se mantiene versionado mediante un repositorio GitHub, permitiendo tener las versiones estables del proyecto y recuperar versiones anteriores en caso de fallos irremediables. Las actualizaciones de dependencias se hacen de forma automática por el archivo requirements.txt que se carga cada vez que se lanza el proyecto.

Como ya he remarcado el uso de Docker simplifica el mantenimiento ya que cualquier modificación del código puede desplegarse de nuevo reconstruyendo la imagen y reiniciando los contenedores sin afectar el entorno base.

# Conclusiones y trabajo futuro.

Este proyecto tenia como objetivo principal aprender la tecnología más puntera del mundo, o lo que es lo mismo, el funcionamiento practico de los modelos de inteligencia artificial. Con el paso del proyecto las expectativas iniciales no solo se han cumplido, sino que se han excedido. Llegando al punto de no solo aprender a nivel básico como montar un agente y dirigirle, sino que he aprendido a crear programas de webscraping automatizado. Este cambio supuso un aumento de la complejidad técnica a la par que mayor valor funcional.

Durante el desarrollo además de decisiones técnicas como la ya dicha también se han identificado limitaciones, especialmente en el comportamiento del modelo de lenguaje, que, en ocasiones, genera respuestas inconsistentes o imposibles de controlar con la precisión buscada. Además de la disminución en el ultimo mes de la capacidad para hacer pruebas con el LLM por la restricción ‘shadow’ y digo ‘shadow’ porque no aparecía en ningún lado oficial que pasaran de las 1000 PPD (Peticiones por día) a 20.

El proyecto ha servido como una experiencia de aprendizaje muy completa. En general, el resultado final es una aplicación funcional que cumple con los objetivos planteados y sienta una buena base para construir encima.

# Trabajo futuro

En un escenario con más tiempo y recursos se podría implementar mejoras en el proyecto como

* Incorporación de nuevas fuentes de eventos, como por ejemplo Dice.com, ra-guide, etc. Consiguiendo así una inclusión total de la mayoría de las fiestas de cada ciudad.
* Mejorar el LLM a uno que tenga respuestas mas consistentes y mayor volumen diario de peticiones.
* Por último, hacer que la aplicación funcionase en móvil de forma correcta haciendo que la sección del filtro sea plegable.

# Relación del proyecto con los módulos del ciclo

Claramente el desarrollo de este proyecto guarda una relación directa con varios de los módulos cursados durante el ciclo, ya que sin los conocimientos adquiridos hubiera sido imposible empezar a diseñar e implementar el proyecto.

Aunque si uno cabe a destacar, seria el módulo de Desarrollo Web Entorno Servidor (DWES). Este ha sido clave para la arquitectura general del proyecto, siendo en el mismo donde he adquirido las bases necesarias para trabajar con el framework Django, que constituye el núcleo del sistema. Este modulo tuve la suerte de aprenderlo de la mano de Jorge Dueñas Lerín, formación que resulto especialmente relevante en mi recorrido estudiantil.

Además de DWES me impartió Despliegue de aplicaciones Web (DAW) donde aprendí los principios de Docker, segundo pilar de la aplicación.

Por último, el lenguaje en el que esta hecha toda la lógica interna de la aplicación, Python. Lenguaje aprendido por primera vez por David Mateos Jiménez. Consiguiendo que me decantase por este lenguaje por encima de los demás.

En conjunto, este proyecto actúa como una integración práctica de los conocimientos adquiridos a lo largo del ciclo, permitiendo concentrar de forma real cada parte de los conceptos teóricos y técnicos vistos durante el ciclo.

# Bibliografía/Webgrafía

**Páginas Web:**

* Chat GPT: <https://chatgpt.com>
* AI studio: <https://aistudio.google.com>
* LangChain: <https://www.langchain.com>
* Django: <https://www.djangoproject.com>
* W3schools: <https://www.w3schools.com>

# Anexos

# Estructura del proyecto.

Se incluye una captura de la estructura de directorios del proyecto para facilitar la comprensión de la organización interna de la aplicación. Habiendo separado cada funcionalidad por app, dejando la app principal ‘ravedaddy’ para que funcione como núcleo de la aplicación.

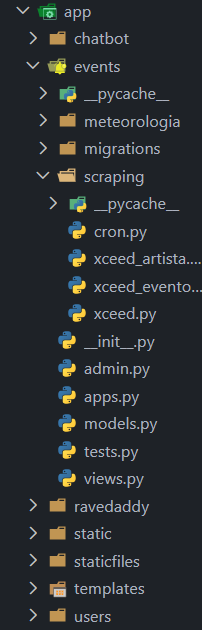


Ilustración 21 Estructura del proyecto

# Ficheros de configuración Docker.

Se adjuntan los principales ficheros de configuración utilizados para la instalación y el despliegue de la aplicación.

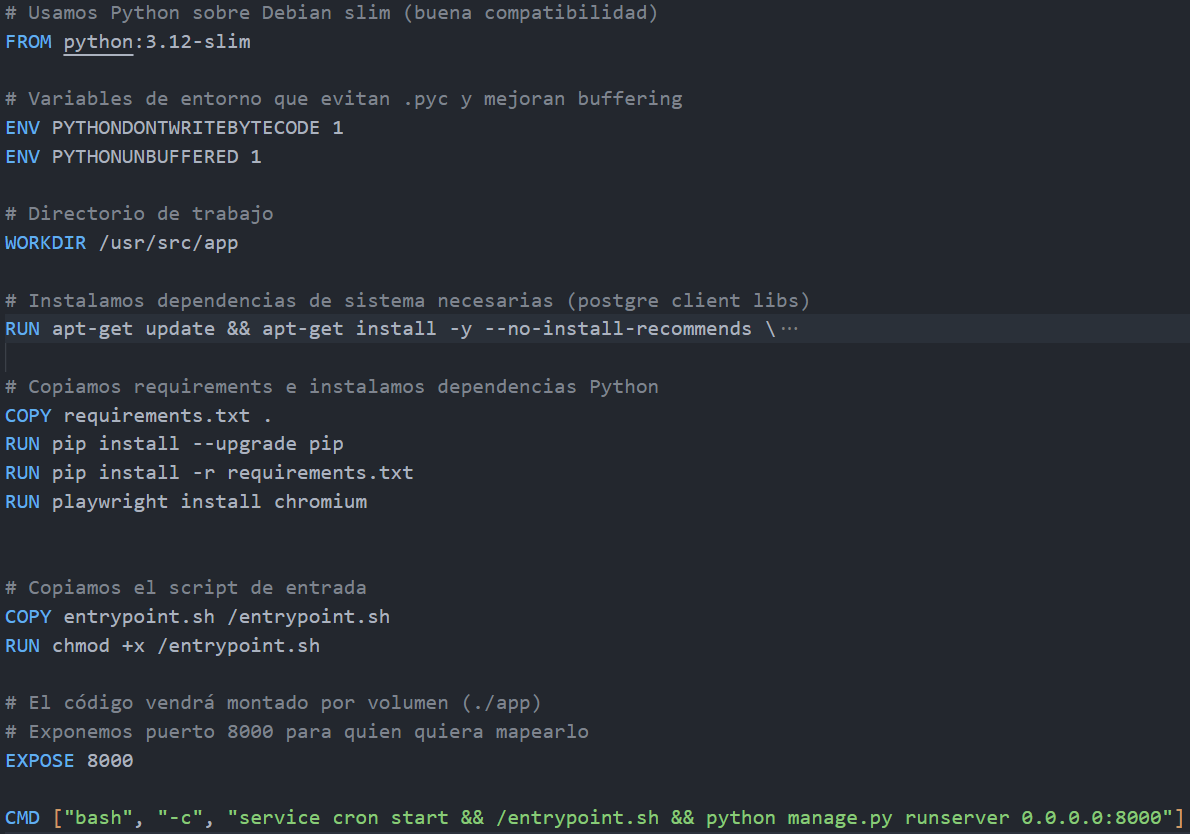


Ilustración 22 Dockerfile

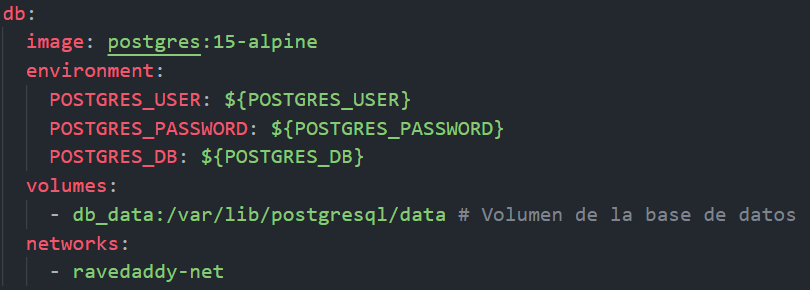


Ilustración 23 Contenedor de la base de datos

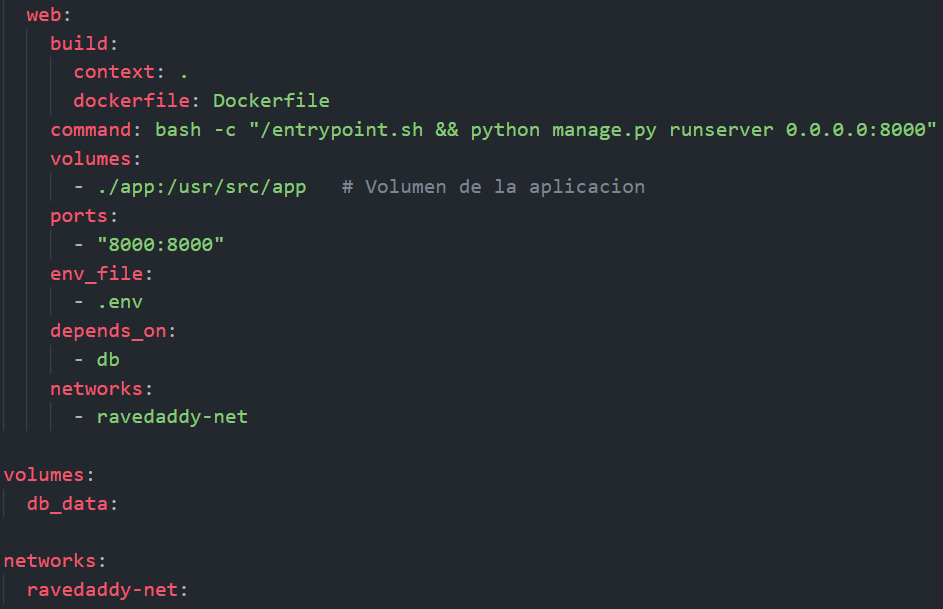


Ilustración 24 Contenedor web