Manual Proyecto Web services

Introducción.

El proyecto de software consiste en un buscador de clima que permite a los usuarios consultar las condiciones meteorológicas en base a su ticket de vuelo,nombre de ciudad y iata. Para ello, utiliza la API de OpenWeather, ofreciendo datos climáticos en tiempo real de manera accesible y precisa. Dado que este tipo de software puede manejar datos sensibles y depende de servicios externos, es fundamental cumplir con una serie de lineamientos generales, tratados internacionales, leyes, códigos y reglamentos. En particular, este análisis se centrará en las normativas aplicables en México y otros marcos legales relevantes.

Lineamientos generales.

-Tratados internacionales

El proyecto se encuentra alineado con diversos tratados internacionales de protección de datos, entre los cuales ocupa un lugar central el Convenio 108 del Consejo de Europa al que se encuentra adherido México. Dicho convenio fija los estándares para la protección de datos, de tal forma que puede ser aplicado a cualquier sistema que maneje información personal, como lo puede ser el buscador de clima, en tanto que recolecta o procesa información sobre las localizaciones de los usuarios o las poblaciones que son objeto de su consulta. Dentro de este panorama, la Ley General de Protección de Datos de la Unión Europea (GDPR) podrá resultar relevante en caso de que el software sea utilizado por ciudadanos de la Unión Europea, dado que regula concretos requerimientos para el manejo de datos personales.En el plano de las normas de protección de datos, el T-MEC, una vez más, también es relevante, en tanto que incorpora normas en la materia, en donde se establecen estándares para la protección de datos para todos los Estados partes, por lo que sí se implementará el proyecto en estos Estados con el fin de asegurar el adecuado manejo de los datos, el mismo deberá cumplir los estándares necesarios de privacidad y las exigencias de seguridad de datos. Finalmente, el Tratado de Comercio Digital del T-MEC impulsa la innovación y el uso responsable de las tecnologías digitales, de tal

forma que asegure que las plataformas online respetan la privacidad y no recolectan información no requerida.

Leyes

La Ley Federal de Protección de Datos Personales en Posesión de los Particulares (LFPDPPP) en México, aunque la legislación en el país es muy escasa, es la ley más relevante en la protección de la privacidad. En la medida en la cual el software puede recoger información acerca de las ciudades consultadas por los usuarios o de su ubicación, es prioritario garantizar que el tratamiento de datos personales siga lo dispuesto en la LFPDPPP. Elementos clave de la LFPDPPP son los siguientes:

Consentimiento expreso del usuario. Cuando el software recoge o almacena datos de carácter personal se debe contar con el consentimiento expreso del usuario, lo que debe hacerse de forma clara y comprensible para el usuario. De este modo, se garantiza que el usuario esté perfectamente informado sobre el uso que la aplicación va a dar a sus datos.

Derechos ARCO (Acceso, Rectificación, Cancelación y Oposición). Los usuarios deben contar con la opción de acceder a los datos que el sistema tiene almacenados sobre ellos, rectificar los datos si son erróneos, solicitar su cancelación o, incluso, oponerse al tratamiento de sus datos en el momento que lo deseen.

Además el software debe ceñirse a la Ley de Protección al Consumidor garantizando la transparencia y la claridad en la prestación de los servicios que ofrece a los usuarios; esto es así dado que el proyecto tiene lugar en la red y puede interactuar con un público amplio de usuarios.

códigos

El desarrollo de software debe seguir los principios establecidos en los **Códigos de Conducta** para garantizar la responsabilidad ética en el manejo de la información. El **Código de Ética Profesional del Ingeniero en Sistemas Computacionales** destaca la importancia de la seguridad y la responsabilidad en la creación de tecnologías digitales. El desarrollador, en este caso, ha implementado buenas prácticas de programación segura, tales como la protección de las credenciales API y el uso de protocolos seguros, para evitar cualquier tipo de exposición de datos o vulnerabilidad en el sistema.

Asimismo, el proyecto respeta el **Código de Comercio Electrónico**, asegurando que la plataforma en línea sea segura, transparente y cumpla con las normativas establecidas para transacciones digitales.

El software depende de la API de OpenWeather para proporcionar datos climáticos. Es crucial que el desarrollador cumpla con las políticas de uso de esta API, que incluyen:

- **Límites de consulta**: Muchas APIs, incluida OpenWeather, imponen límites sobre la cantidad de solicitudes que se pueden realizar en un periodo determinado. El software debe respetar estos límites para evitar la interrupción del servicio.
- **Licencias de uso**: Es necesario que el desarrollador se asegure de tener las licencias adecuadas, especialmente si el software se implementa a gran escala.
- **Atribución de la fuente**: Muchas APIs exigen que se reconozca la fuente de los datos. En este caso, el software debe mostrar de manera visible que la información meteorológica proviene de OpenWeather.

Reglamentos

En términos de reglamentación específica, el **Reglamento de la Ley Federal de Protección de Datos Personales en Posesión de los Particulares** establece obligaciones claras para quienes manejan datos personales, como la necesidad de implementar políticas de privacidad transparentes. Aunque el software no almacena información personal sensible de forma predeterminada, se han adoptado medidas para garantizar que cualquier dato que se procese cumpla con los estándares legales de protección de datos.

Además, el **Reglamento del Instituto Federal de Telecomunicaciones (IFT)** exige que se proteja la confidencialidad y la seguridad de los datos transmitidos a través de redes de telecomunicaciones. El uso de protocolos seguros como HTTPS para la transmisión de datos entre el software y la API de OpenWeather es un aspecto clave para asegurar la integridad de la información y evitar que sea interceptada o modificada durante su transferencia.

Generalidades

Estructura y arquitectura del proyecto.

Estructura del paquete del proyecto.

- weather_app/
  - static/
    - IATAS.csv
    - IATAS.json
    - data.set
    - favicon.io
    - styles.css
    - metodos.js
    - hist.js
    - script.js
    - sugerencias.js
  - templates/
    - historial.html
    - index.html
  - test/
    - __init.py__
    - conftest.py
    - test_exceptions.py
    - test_iatas.py
    - test_prediccion.py
  - utils/
    - __init.py__
    - cacheyEscritura.py
    - horasyTiempo.py
    - iatas.py
    - obtenerDatosFINALES.py
    - obtenerclimas.py
    - obtenervuelos.py
    - predicc.py
    - traductor.py

        - __init.py__
        - app.py
        - install.py
        - requirements.txt
        - wsgi.py
    - README.md

Arquitectura general.

La aplicación sigue una arquitectura cliente-servidor, con una clara división de responsabilidades entre el backend y el frontend.

Backend: Desarrollado con el lenguaje de programación Python,en especial utilizando el framework Flask, el cual sigue de manera flexible el patrón de diseño MVC (Modelo-Vista-Controlador). Esta estructura modular facilita el desarrollo de aplicaciones sin la necesidad de depender de bibliotecas externas. El backend se encarga de gestionar los datos, procesar las solicitudes y proporcionar las respuestas necesarias al frontend.

Frontend: Desarrollado con JavaScript, permite una interacción dinámica con el usuario, mejorando la experiencia y comprensión de la interfaz. Además, utiliza HTML y CSS para diseñar y estructurar la interfaz de usuario. El frontend se encarga de realizar peticiones HTTPS al backend, recibir las respuestas y mostrarlas de manera efectiva al usuario.

Dependencias externas.

El programa utiliza dos APIs principales. La primera, llamada "OpenWeather", se encarga de recopilar datos climáticos según los parámetros definidos por el algoritmo del programa. La segunda API, "Aviationstack", se utiliza para obtener información sobre vuelos a través de los códigos IATA de cada vuelo que se quiere consultar. Esta API proporciona detalles importantes como la ubicación de origen y destino, la hora de despegue y llegada, información crucial para consultar las condiciones climáticas en los lugares involucrados en el vuelo.
También, el programa depende de módulos de python externos los facilitan el flujo del algoritmo que administra el backend. Los cuales son:

- beautifulsoup4==4.12.3

- blinker==1.8.2
- certifi==2024.8.30
- chardet==3.0.4
- charset-normalizer==3.3.2
- click==8.1.7
- deep-translator==1.11.4
- Flask==3.0.3
- fuzzywuzzy==0.18.0
- googletrans==4.0.0rc1
- h11==0.9.0
- h2==3.2.0
- hpack==3.0.0
- hstspreload==2024.9.1
- httpcore==0.9.1
- httpx==0.13.3
- hyperframe==5.2.0
- idna==2.10
- iniconfig==2.0.0
- itsdangerous==2.2.0
- Jinja2==3.1.4
- Levenshtein==0.25.1
- MarkupSafe==2.1.5
- numpy==2.1.0
- packaging==24.1
- pandas==2.2.2
- pluggy==1.5.0
- pytest==8.3.2
- python-dateutil==2.9.0.post0
- pytz==2024.1
- rapidfuzz==3.9.6
- requests==2.32.3
- rfc3986==1.5.0
- six==1.16.0
- sniffio==1.3.1
- soupsieve==2.6
- tzdata==2024.1
- urllib3==2.2.2
- Werkzeug==3.0.4

Entorno de desarrollo.

Para gestionar eficazmente las dependencias en el desarrollo de la aplicación, se optó por crear un entorno virtual de Python utilizando la herramienta venv. Este enfoque permite aislar las dependencias del proyecto, evitando conflictos con otras aplicaciones y garantizando una ejecución consistente.

Dentro de este entorno virtual, todas las dependencias necesarias se enumeraron en un archivo denominado requirements.txt. Este archivo sirve como una lista detallada de los paquetes y versiones específicos que la aplicación requiere, facilitando su instalación y actualización de manera ordenada.

Este método no solo asegura que el entorno de desarrollo sea reproducible en diferentes máquinas, sino que también simplifica la gestión y mantenimiento de las dependencias a lo largo del ciclo de vida del proyecto.

Además, se desarrolló un módulo de Python llamado install.py, el cual automatiza la creación del entorno virtual y la instalación de todas las dependencias necesarias del proyecto. Este módulo se encarga de configurar el entorno de ejecución de forma automática, facilitando el proceso de instalación para los usuarios. Además, install.py también incluye la funcionalidad para ejecutar la aplicación desarrollada con Flask, simplificando el despliegue del proyecto y su puesta en marcha con un solo comando.

¿Cuál es el público objetivo del software?

El software está diseñado para ser utilizado por diferentes tipos de usuarios, cada uno con roles y responsabilidades específicas dentro del sistema. A continuación, se describe a los principales grupos a quienes está dirigido:

- **Usuarios finales**:
  Son los usuarios generales del sistema, que interactúan principalmente con la interfaz de usuario para acceder a las funcionalidades del software.
    - **Objetivos**: Los usuarios finales buscan realizar tareas específicas dentro del sistema, como acceder a información del clima de un lugar en específico o de un vuelo en específico.

    - **Experiencia de usuario**: El software está diseñado para ser intuitivo y fácil de usar, con una interfaz amigable y accesible para usuarios de todos los niveles técnicos.

- **Administradores de contenido**:
  Son los responsables de gestionar el contenido que se presenta en la aplicación o sistema. Este grupo tiene acceso a funcionalidades avanzadas para crear, editar y eliminar contenido de la plataforma, como noticias, artículos, productos, entre otros.
    - **Objetivos**: Mantener actualizado el contenido y asegurar que la información presentada sea relevante y precisa.
- **Webmasters (Administradores técnicos)**:
  Este grupo tiene un control completo sobre el sistema, incluyendo la gestión de la información sensible en el proyecto. Son los encargados de mantener el sistema en funcionamiento, implementar actualizaciones y asegurar la seguridad del software.
    - **Objetivos**: Garantizar la estabilidad, seguridad y correcto funcionamiento del sistema.
    - **Acceso y funcionalidades**: Tienen acceso a todas las funcionalidades del sistema, incluyendo paneles de administración avanzados y herramientas de monitoreo y depuración.
- **Desarrolladores o equipo técnico**:
  Aunque no son usuarios finales del software, los desarrolladores están encargados de crear, mantener y mejorar el sistema. Esto incluye la implementación de nuevas características, corrección de errores y optimización de rendimiento.
    - **Objetivos**: Asegurar que el software cumpla con los requerimientos funcionales y no funcionales del proyecto, mientras mantienen una base de código eficiente y bien estructurada.

Información de software.

Tipo de software.

Se trata de una aplicación web diseñada para la consulta en tiempo real del clima en ciudades específicas, ya sea para obtener información meteorológica de una ciudad o región en particular relacionada con un vuelo. Esta aplicación utiliza servicios web como OpenWeatherMap para acceder a datos meteorológicos y está enfocada en ser una herramienta práctica para sobrecargos, pilotos y clientes que requieren conocer las condiciones climáticas en los aeropuertos de origen y destino.

Requerimientos.

Requisitos Funcionales.

- **Consulta de Clima en Tiempo Real**
  La aplicación es capaz de realizar peticiones a las APIs de OpenWeather y AviationStack en tiempo real, proporcionando al usuario la información meteorológica más actualizada según sus necesidades.
- **Entrada de Datos**
  El usuario puede consultar el clima mediante dos opciones de búsqueda:

  Búsqueda por ciudad o código IATA del aeropuerto

  Permite al usuario ingresar el nombre de la ciudad donde se ubica el aeropuerto o su código IATA para obtener las condiciones climáticas de dicho lugar.

  Búsqueda por ticket de vuelo

  Se realiza a través del código IATA del vuelo, ya que este contiene información detallada sobre las horas y los aeropuertos de origen y destino del vuelo. Esta información es esencial para consultar el clima de las ubicaciones relacionadas con el itinerario del vuelo.

Validación de Datos

Para evitar errores en las consultas, se han implementado métodos de verificación de datos, asegurando que las entradas proporcionadas por el usuario sean correctas y válidas antes de procesar las solicitudes.

- **Despliegue de Información del Clima**
  La aplicación presenta la información meteorológica de manera clara y concisa, con un diseño intuitivo y fácil de entender, lo que facilita su uso para cualquier persona que desee consultar el clima de una región específica.
- **Gestión de Entradas Flexibles**
  La aplicación es capaz de manejar ligeros errores ortográficos en las búsquedas por ciudad o código IATA, lo que mejora la experiencia del usuario. Las consultas de clima están limitadas a lugares relacionados con el Aeropuerto Internacional de la Ciudad de México, permitiendo que los usuarios obtengan información solo de los destinos donde dicho aeropuerto opera despegues y aterrizajes. Esto asegura que las búsquedas estén orientadas a destinos relevantes y autorizados por el aeropuerto.
- **Manejo de Errores**
  El sistema está diseñado para tolerar y manejar diferentes tipos de errores, como la introducción de datos incorrectos o fallos en las APIs. Esto permite que el usuario reciba mensajes claros sobre el tipo de error que está ocurriendo, mejorando la experiencia de uso y facilitando la identificación y solución de problemas.
- **Almacenamiento de Caché**
  La aplicación cuenta con la capacidad de almacenar en caché las consultas realizadas por el usuario, lo que optimiza el rendimiento al evitar consultas repetidas de la misma información. Para garantizar que los datos mostrados sean actuales, el sistema elimina automáticamente el caché que contiene información desactualizada. Además, se incluye un historial donde los usuarios pueden revisar las consultas realizadas anteriormente, lo que les permite acceder fácilmente a información previa sin tener que realizar nuevas solicitudes.

Requisitos No funcionales.

- **Rendimiento**

  La aplicación ofrece un tiempo de respuesta considerablemente rápido, aunque también depende de la velocidad de las respuestas proporcionadas por las APIs externas utilizadas en el proyecto. El procesamiento interno de los datos y su transmisión al frontend son eficientes, minimizando el tiempo de espera para el usuario. En general, la aplicación

proporciona respuestas en un tiempo cómodo, garantizando una experiencia de usuario fluida y satisfactoria.

- **Disponibilidad**

  La aplicación presenta una limitación en cuanto al número de consultas climáticas debido a las restricciones impuestas por las APIs externas. Sin embargo, esta limitación se mitiga mediante el uso de caché, lo que permite reutilizar la información almacenada previamente en el sistema, evitando la repetición de consultas sobre la misma información. Esto optimiza la disponibilidad de las consultas para el usuario. Además, la aplicación implementa un mecanismo anti-spam, que inhabilita la posibilidad de realizar nuevas búsquedas por unos segundos tras una consulta climática. Esta funcionalidad mejora la eficiencia del sistema y garantiza la disponibilidad continua de información sin sobrecargar las APIs.

- **Usabilidad**

  Este proyecto cuenta con una interfaz de usuario intuitiva y fácil de utilizar. El diseño está estructurado de manera ordenada y organizada, lo que permite al usuario comprender rápidamente cómo interactuar con la aplicación. Cada elemento de la interfaz está diferenciado visualmente, utilizando colores y fuentes únicos para evitar confusiones al interactuar con los distintos componentes.

  Además, el diseño es completamente **responsive**, lo que garantiza que la información se muestre de manera clara y adaptada, sin importar el dispositivo que el usuario esté utilizando para consultar el clima, ya sea un teléfono móvil, una tableta o un ordenador. Esto asegura una experiencia de usuario óptima en cualquier plataforma.

- **Compatibilidad y Portabilidad**

  El programa ha sido diseñado como una aplicación web, lo que permite su ejecución en cualquier dispositivo que tenga un navegador web, lo que garantiza una amplia compatibilidad con diferentes plataformas. Esta característica significa que los usuarios pueden acceder a la aplicación desde ordenadores, tabletas y teléfonos móviles, sin necesidad de instalaciones adicionales.

Además, la implementación de la aplicación es ligera, lo que implica que no requiere muchos recursos del sistema para funcionar. Esto la hace ideal para dispositivos con especificaciones de hardware y software estándar, ya que puede operar eficientemente sin comprometer el rendimiento. En resumen, la aplicación es accesible y fácil de usar en una variedad de dispositivos, lo que mejora la experiencia del usuario.

- **Seguridad**

  El sistema ha sido diseñado para verificar que los datos ingresados por el usuario cumplan con el formato requerido antes de realizar una consulta a las APIs y mostrar la información. Además, se implementa una restricción que impide consultas repetitivas en cortos periodos de tiempo, evitando sobrecargas innecesarias.

  Por otro lado, el uso de consultas HTTPS garantiza la transferencia segura de datos, asegurando que la comunicación entre el backend y el frontend esté protegida contra vulnerabilidades. Esto asegura que la información transmitida no pueda ser interceptada o manipulada por terceros.

  En conjunto, estas medidas crean un sistema robusto y seguro que cubre las principales vulnerabilidades potenciales de una aplicación web, ofreciendo una mayor confianza en la protección de los datos y la integridad del sistema.

Enfoque

El proyecto está enfocado en la consulta de climas tanto de la ciudad de salida como de la ciudad de llegada, pero exclusivamente para vuelos que parten o llegan al Aeropuerto Internacional de la Ciudad de México (AICM). Las consultas se realizan para vuelos que salen el mismo día en que se ejecuta el algoritmo, y solo para destinos a los que el AICM tiene rutas activas.

Este sistema está dirigido a sobrecargos, pilotos y clientes que no necesariamente tienen conocimientos técnicos en programación. Por eso, ofrece una presentación de la información de manera clara y sencilla para los usuarios comunes, mientras que para sobrecargos y pilotos incluye detalles técnicos adicionales que son relevantes para sus necesidades operativas.

Diagramas de Flujo y PseudoCódigo del funcionamiento del sistemas.

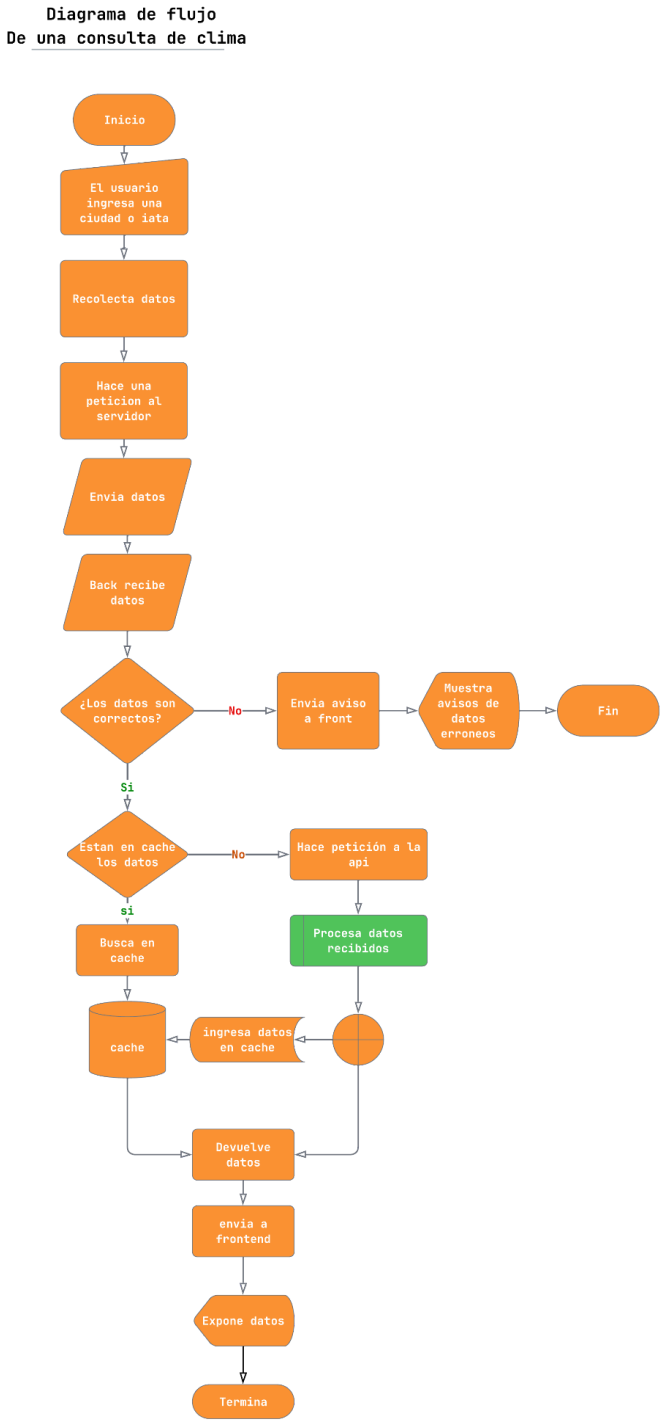


Diagrama de flujo

Procesamientos de datos de una cosulta por ciudad-iata

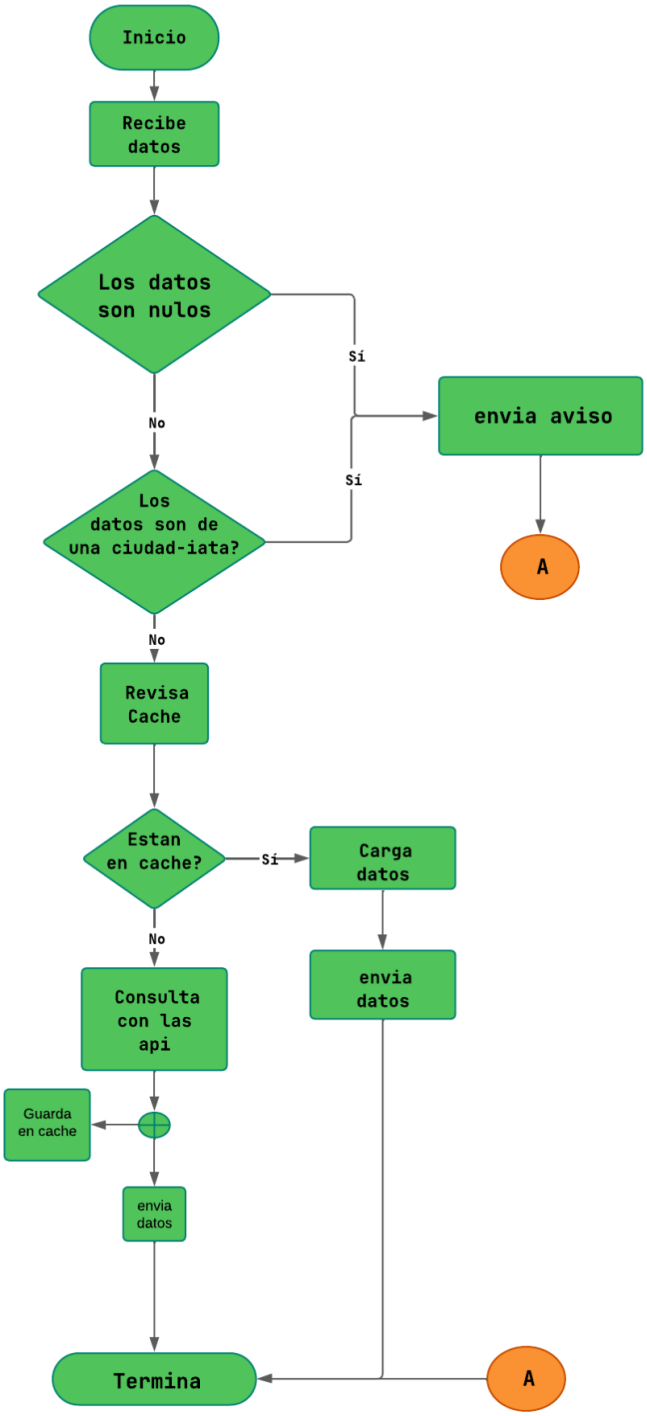
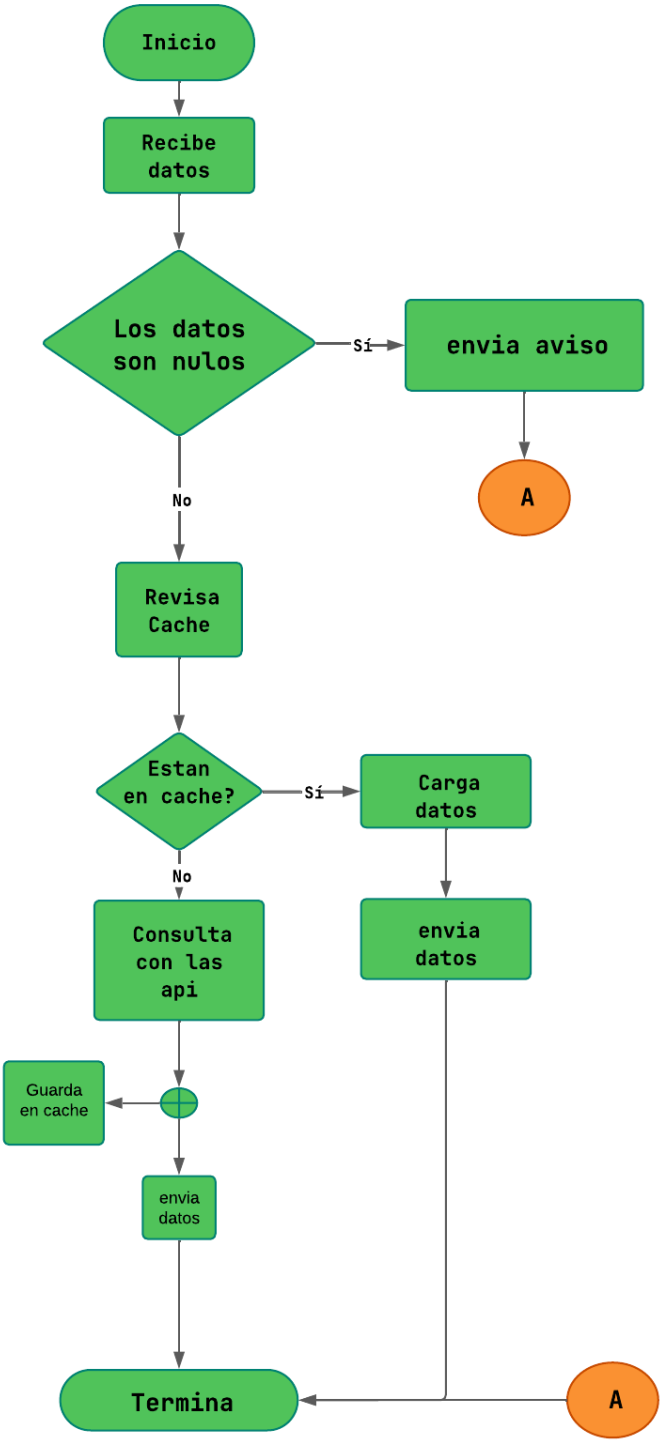


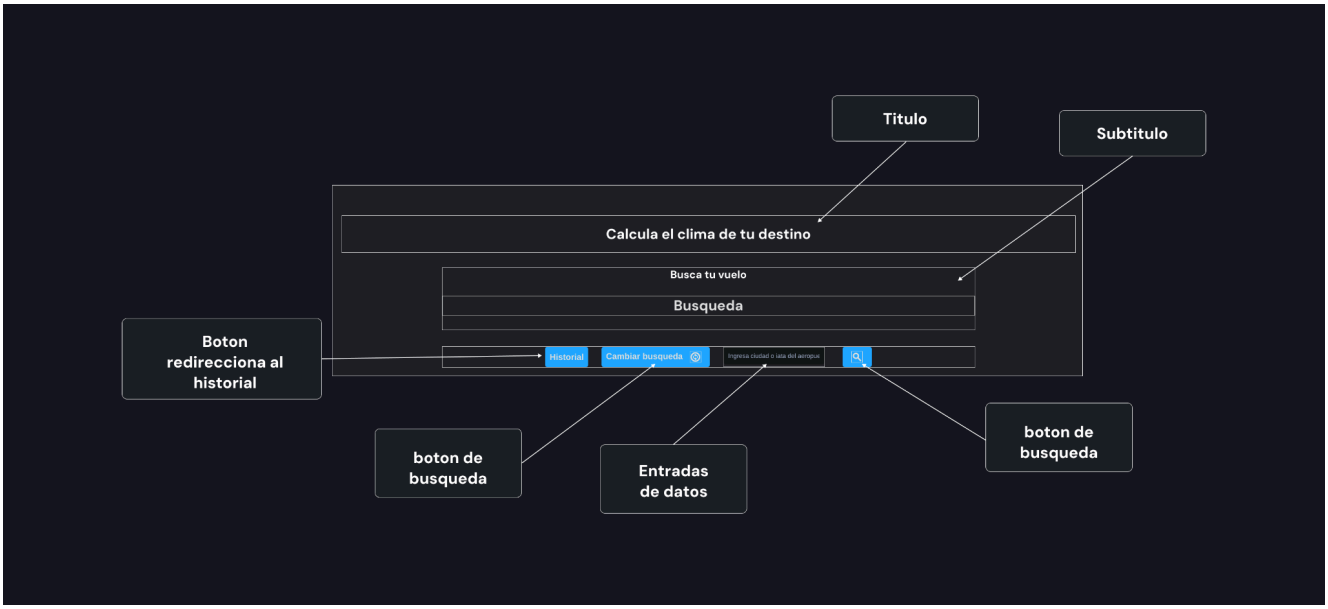
Diagrama de flujo

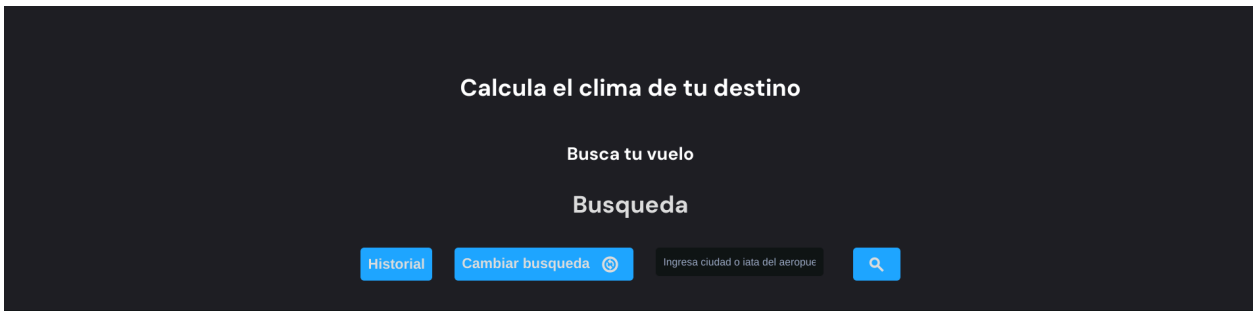
Procesamientos de datos de una cosulta por ticket



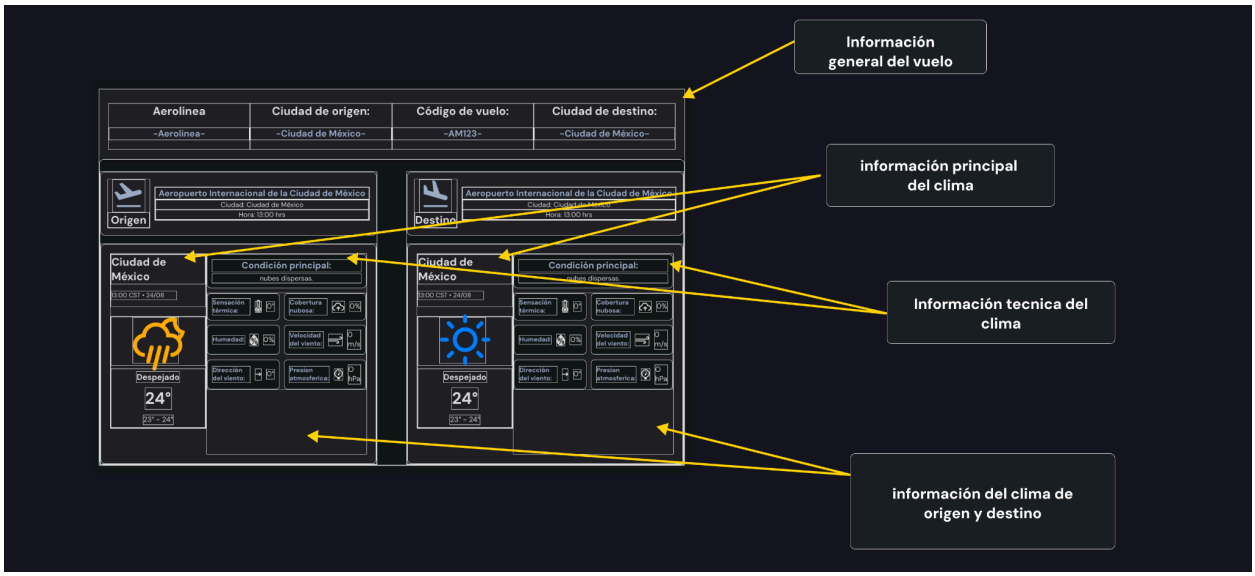
Diseño de la interfaz.

- Área de búsqueda.

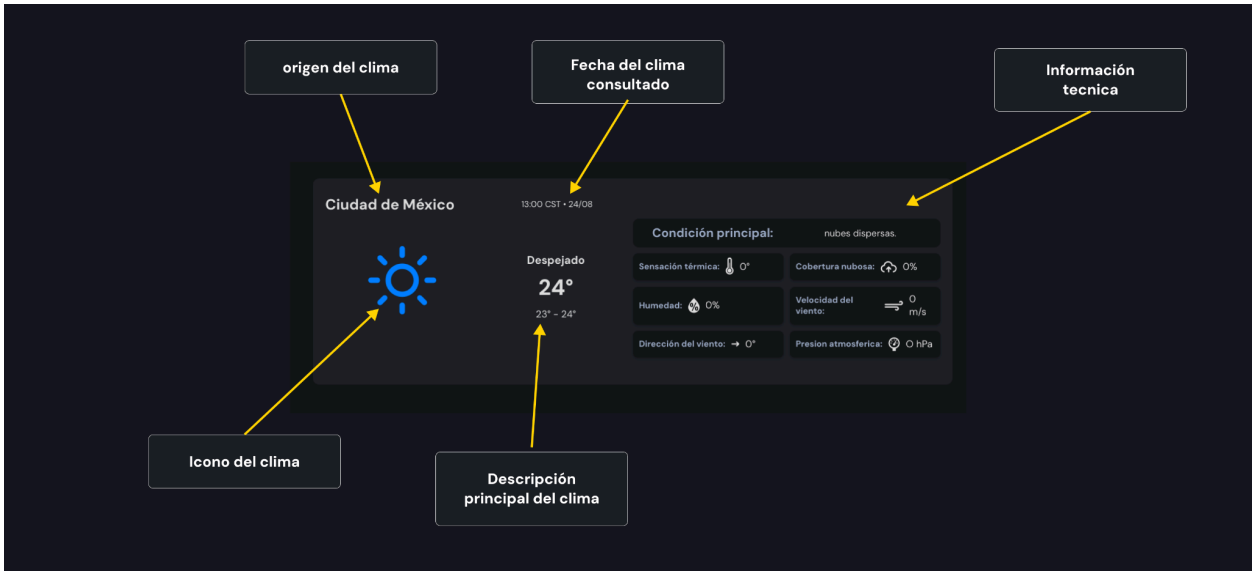




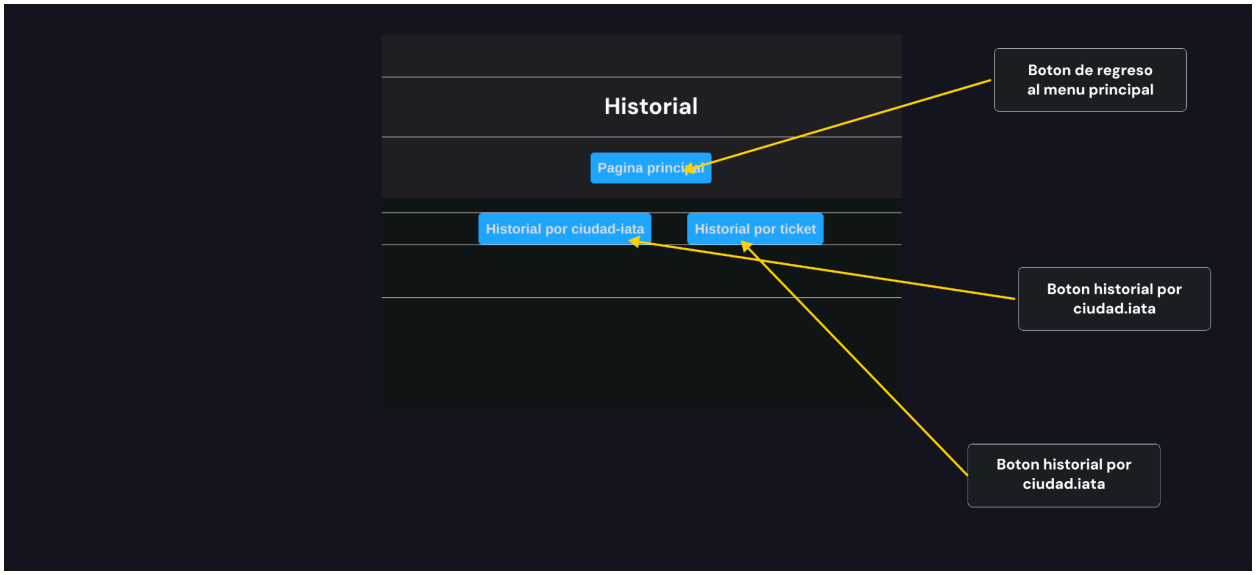
- Cuadro de información de búsqueda por ticket

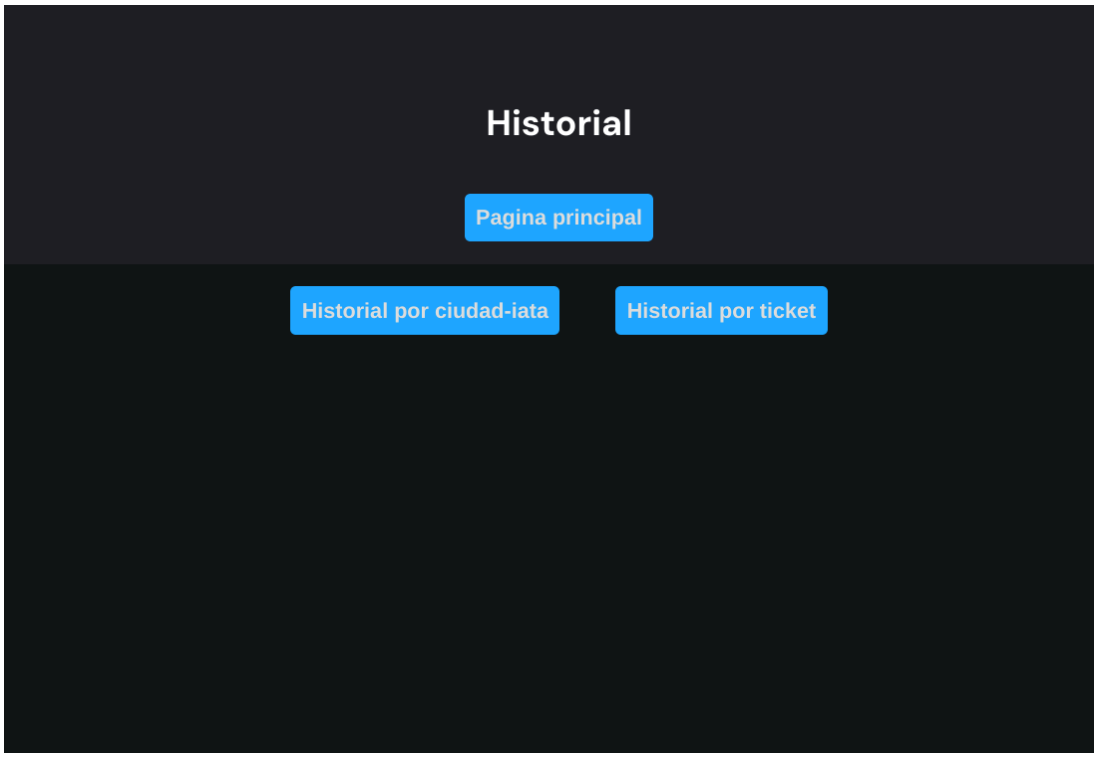


- Cuadro de información de búsqueda por Ciudad-iata



- Área de Historial.





Paleta Colores y Tipografía

Paleta de colores

Botones de interacción
- #1DA7FF
- 29, 167, 255
- rgb(29, 167, 255)
- rgba(29, 167, 255, 1)

Botones de interacción al hacer focus
- #242933
- 36, 41, 51
- rgb(36, 41, 51)
- rgba(0, 85, 178, 1)

Fondo principal

- #121416
- 18, 20, 22
- rgb(18, 20, 22)
- rgba(18, 20, 22, 1)

Fondo secundario
- #1D1E24
- 29, 30, 36
- rgb(29, 30, 36)
- rgba(29, 30, 36, 1)

Color de letra principal
- #FAFAFA
- 250, 250, 250
- rgb(250, 250, 250)
- rgba(250, 250, 250, 1)

Color de letra secundario
- #9BAAC4
- 155, 170, 196
- rgb(155, 170, 196)
- rgba(155, 170, 196, 1)

Color de letra de botones e información
- #E0E0E0
- 224, 224, 224
- rgb(224, 224, 224)
- rgba(224, 224, 224, 1)

Icono dia
- #FFAD00
- 255, 173, 0
- rgb(255, 173, 0)
- rgba(255, 173, 0, 1)

Icono noche
- #0080FF
- 0, 128, 255
- rgb(0, 128, 255)
- rgba(0, 128, 255, 1)

Tipografía:

La tipografía usada en este proyecto fue DM Sans.

DM Sans
DM Sans
DM Sans
DM Sans
DM Sans
DM Sans
DM Sans
DM Sans
DM Sans

Procedimientos.

Mantenimiento

Actualización de dependencias:

- **Monitoreo de versiones:** Utilizar herramientas como pip-tools o dependabot para rastrear actualizaciones de bibliotecas externas como deep_translator, FuzzyWuzzy, y Flask. Cada vez que haya una nueva versión, se deben revisar los *release notes* para identificar cambios de compatibilidad, mejoras de rendimiento o correcciones de seguridad.
- **Pruebas de regresión:** Antes de actualizar cualquier dependencia, realizar pruebas de regresión exhaustivas para asegurarse de que las actualizaciones no rompan funcionalidad existente.
- **Gestión de dependencias con entornos virtuales:** Actualizar el archivo requirements.txt o Pipfile para reflejar las versiones más recientes y crear un entorno virtual limpio. Validar la correcta instalación de todas las dependencias.

Optimización del código:

- **Refactorización modular:** Si algunas funciones o clases en app.py y otros módulos están manejando demasiadas responsabilidades, reorganizar el código en módulos más pequeños y cohesionados. Esto hará que el proyecto sea más mantenible y escalable a largo plazo.
- **Uso eficiente de recursos:** Revisar cómo se manejan las consultas a las APIs y la memoria. Por ejemplo, si la aplicación realiza múltiples consultas a APIs para una misma ciudad dentro de un corto periodo, se podría usar un sistema de caché (e.g., Redis) para reducir la cantidad de consultas externas.
- **Eliminación de código redundante:** Si el proyecto ha crecido de manera incremental, puede haber código duplicado o innecesario. Revisar el proyecto para eliminar redundancias y simplificar funciones.

Mejora del manejo de errores:

- **Establecer un middleware de manejo de errores:** En Flask, crear un middleware que capture todas las excepciones y errores que ocurran en las rutas, devolviendo respuestas JSON bien estructuradas y con códigos HTTP adecuados (e.g., 400 para errores del cliente, 500 para errores del servidor).
- **Logs y monitoreo:** Implementar un sistema de registro de errores (logging) más robusto. Usar bibliotecas como Sentry o Loggly para capturar errores en tiempo real y monitorear el estado de la aplicación en producción.
- **Tiempo de respuesta de APIs:** Si una API tarda demasiado en responder, establecer límites de tiempo (timeouts) y mecanismos de reintento con backoff exponencial. Esto evitará que la aplicación quede bloqueada esperando una respuesta indefinida.

Ampliación de pruebas automatizadas:

- **Cobertura completa de tests:** Aumentar la cobertura de tests unitarios e integrar herramientas como pytest-cov para medir el nivel de cobertura del código. Asegurarse de probar diferentes escenarios de entrada (ciudades inexistentes, respuestas vacías de la API, nombres de ciudades mal escritos).
- **Pruebas de integración con APIs:** Implementar pruebas que simulen la interacción real con las APIs, usando mocks para evitar depender de conexiones a Internet durante los tests.

- **Tests de carga y rendimiento:** Usar herramientas como Locust o JMeter para ejecutar pruebas de carga que simulen múltiples usuarios consultando el clima simultáneamente, identificando posibles cuellos de botella en el rendimiento.

Mejora de la documentación:

- **Comentarios en el código:** Asegurarse de que cada función y módulo tengan comentarios que expliquen el propósito de la función, los parámetros que reciben y los resultados que devuelven, sin necesidad de ejemplos redundantes.
- **Documentación externa:** Crear una guía para nuevos desarrolladores que incluya cómo configurar el entorno, cómo ejecutar los tests, y una descripción general del flujo de la aplicación. Usar herramientas como Sphinx para generar documentación técnica a partir del código.
- **Manual de uso:** Incluir un archivo README.md detallado con ejemplos de cómo interactuar con las rutas de la API, incluyendo ejemplos de solicitudes y respuestas esperadas.

Seguridad:

- **Gestión de claves API:** Asegurarse de que las claves de las APIs se manejen de forma segura, utilizando variables de entorno almacenadas en archivos .env, en lugar de hardcodearlas en el código fuente. También considerar herramientas como AWS Secrets Manager o Vault para manejar claves de acceso en un entorno de producción.
- **Validación de entradas:** Implementar una validación más estricta de las entradas de los usuarios. Por ejemplo, si los usuarios introducen nombres de ciudades, asegurarse de que no haya inyecciones de SQL, XSS u otros tipos de ataques.
- **Rate Limiting:** Implementar limitaciones de tasa (*rate limiting*) para prevenir que usuarios maliciosos realicen múltiples solicitudes a la API en cortos periodos de tiempo, evitando posibles ataques de denegación de servicio (DDoS).

Actualizaciones

Interfaz de usuario (UI):

Diseño responsivo: Incorporar un diseño que se ajuste automáticamente a diferentes tamaños de pantalla (dispositivos móviles, tablets y computadoras) utilizando CSS moderno o frameworks como Bootstrap o TailwindCSS. Esto mejoraría la experiencia de los usuarios que acceden desde dispositivos móviles.

- **Mejora de la experiencia de usuario (UX):** Implementar un sistema de sugerencias mientras el usuario escribe el nombre de la ciudad (autocompletado), mostrando posibles opciones a medida que escribe. Esto agilizaría la búsqueda y reduciría errores de entrada.
- **Sugerencias según el estado del clima:**Poder hacer pequeñas recomendaciones al usuario según el clima de la ciudad consultada.

Internacionalización (i18n):

- **Ampliación de idiomas:** Aumentar el número de idiomas compatibles mediante deep_translator, permitiendo que el sistema detecte el idioma del navegador del usuario y muestre el contenido en ese idioma de forma automática.
- **Traducción dinámica:** Implementar un selector de idioma en la interfaz que permita a los usuarios cambiar el idioma de la página sin recargarla, utilizando técnicas de traducción dinámica mediante JavaScript.

Optimización de consultas climáticas:

- **Caché distribuido:** Usar herramientas como Redis o Memcached para almacenar en caché las respuestas de consultas climáticas por un periodo de tiempo configurable (e.g., 10 minutos). Esto reduciría la latencia y la carga sobre las APIs externas.
- **Balanceo de carga entre APIs:** Si el proyecto soporta múltiples APIs climáticas, implementar un balanceador de carga inteligente que distribuya las solicitudes entre las diferentes APIs disponibles, seleccionando la API más rápida o la que tenga mejor disponibilidad en ese momento.
- **Predicciones meteorológicas:** Añadir funcionalidad para mostrar no solo el clima actual, sino también pronósticos

a corto y largo plazo (5 días, 10 días), lo cual podría obtenerse de APIs adicionales o comprando servicios premium.

Autenticación y permisos:

- **Sistema de autenticación:** Implementar OAuth o un sistema de autenticación mediante tokens JWT para permitir a los usuarios iniciar sesión, acceder a consultas previas, y guardar búsquedas frecuentes.
- **Roles de usuario:** Introducir diferentes niveles de acceso, como administradores que pueden modificar la configuración del sistema y usuarios regulares que solo pueden consultar el clima.

Integración con nuevas APIs:

- **Calidad del aire y alertas meteorológicas:** Añadir soporte para consultar datos adicionales, como la calidad del aire, alertas meteorológicas (e.g., huracanes, tormentas), o estadísticas de largo plazo sobre precipitaciones y temperaturas.
- **Geocodificación y búsqueda avanzada:** Implementar una API de geocodificación como Google Maps API para que los usuarios puedan buscar ciudades por coordenadas geográficas (latitud/longitud) en lugar de nombres.

Mejora de la usabilidad:

- **Autocompletado con API de geolocalización:** Integrar una API de autocompletado de ciudades, permitiendo a los usuarios seleccionar ciudades de una lista basada en su geolocalización o preferencias.
- **Búsqueda por región:** Implementar la búsqueda por países o regiones, permitiendo a los usuarios obtener el clima para todas las ciudades dentro de una región geográfica, facilitando las consultas globales.