



# **Manual**

## **Weather Airport**



**Diaz Quijada Alan Joseph**

**Vargas Navarro Lizbeth Montserrat**

**Vega Alonso Diego Hazael**

## **Introducción**

Este proyecto ha sido desarrollado con el objetivo de proporcionar una aplicación web que permita a los usuarios consultar el estado del clima en tiempo real para una variedad de ciudades. La aplicación está diseñada para ser utilizada por diferentes tipos de usuarios, como turistas, sobrecargos y pilotos, brindando datos específicos y relevantes según las necesidades de cada uno.

La aplicación hace uso de servicios externos para la consulta del clima y permite a los usuarios generar tickets de vuelo para realizar búsquedas personalizadas. Asimismo, se integra una interfaz amigable que facilita el uso de la aplicación, adaptándose a las características y preferencias de cada usuario, ya sea mediante la consulta directa por ciudad o a través de un ticket.

El propósito principal de este proyecto es ofrecer una herramienta eficiente y sencilla que permita acceder a información climatológica y de vuelos en tiempo real, con la flexibilidad de adaptarse a distintos tipos de usuarios. Este sistema puede ser útil para el aeropuerto, agencias de viajes y cualquier otro contexto donde la información actualizada sobre el clima y vuelos sea relevante.

## **Lineamientos Generales**

Este proyecto se desarrolla cumpliendo con una serie de leyes y reglamentos que garantizan la protección de los datos personales de los usuarios, la transparencia en el manejo de la información y el cumplimiento de las normas internacionales sobre seguridad y privacidad. A continuación, se describen algunos de los principales lineamientos, leyes y regulaciones aplicables:

### **1. Ley Federal de Protección de Datos Personales en Posesión de los Particulares (México)**

Esta ley tiene como objetivo proteger los datos personales de los individuos y regular su tratamiento por parte de las entidades privadas. En el contexto de este proyecto, se garantiza que la información recopilada de los usuarios, como datos relacionados con sus vuelos y condiciones climáticas, se maneje con total confidencialidad y seguridad. Los datos solo serán utilizados para los fines establecidos.

Los principios de protección de datos que se aplican incluyen:

**Finalidad:** Los datos solo serán utilizados para los fines indicados en la aplicación, como consultas climatológicas o gestión de tickets.

**Acceso, Rectificación, Cancelación y Oposición (ARCO):** Se asegura el derecho de los usuarios a acceder, rectificar o eliminar sus datos personales.

Aunque el proyecto está enfocado en México, es importante tener en cuenta las normativas internacionales que impactan la protección de datos y el uso de servicios en línea.

### **2. Reglamento General de Protección de Datos (GDPR - Unión Europea)**

Esta normativa de la Unión Europea protege los datos personales de los ciudadanos europeos. Requiere el consentimiento explícito del usuario para la recopilación de datos y otorga a los usuarios derechos como el acceso, rectificación, cancelación y portabilidad de sus datos. Si el proyecto se llega a extender a países europeos, será necesario cumplir con este reglamento que regula la privacidad y protección de datos en la Unión Europea.

#### 5. Reglamento de la Ley de Aviación Civil (México)

Dado que el proyecto involucra la consulta de vuelos, se tiene en cuenta el Reglamento de la Ley de Aviación Civil, que regula las actividades aéreas en México. En particular, se respetan las normativas relacionadas con la transparencia de la información de vuelos y el manejo adecuado de los datos de los pasajeros.

#### 7. Decretos y Reglamentos Locales

Dependiendo de la región o país donde se despliegue la aplicación, también se cumplirán con los decretos y regulaciones locales específicas sobre protección de datos y transparencia.

## Generalidades

### Estructura:

El proyecto está organizado de manera modular para facilitar su mantenimiento y escalabilidad.

A continuación se describe la estructura principal del proyecto:

-app/services: Contiene los módulos clave que gestionan los servicios principales del proyecto, como la consulta de vuelos y el clima. Estos módulos interactúan con la API externa y procesan la información necesaria para las funcionalidades del sistema. Algunos de los módulos principales son:

- api\_helpers.py: Módulo encargado de gestionar las llamadas a la API externa.
- weather\_service.py: Proporciona funciones para consultar el clima en diferentes ciudades.
- Robusty\_Entry.py: Trabaja la entrada del usuario, de tal forma que nos puede dar la el nombre de la ciudad y/o la IATA.
- Available\_Flights.py: Gestiona la información sobre vuelos.

-static/css: Contiene los archivos de estilos CSS que definen el aspecto visual de la aplicación. Aquí se encuentran estilos personalizados para botones y otras interfaces.

- styles.css: Define los estilos visuales, como colores, fuentes y disposición de los elementos visuales en la interfaz de usuario.

-templates: Incluye las plantillas HTML que estructuran las páginas visibles para el usuario. Utiliza el sistema de plantillas para integrar dinámicamente los datos en las vistas.

- base.html: Plantilla base que define la estructura principal del sitio web.
- index.html: Página principal donde los usuarios pueden consultar el clima en función de su ticket o ingresando una ciudad.
- ticket.html, ticket\_result.html, ticket\_search.html: Plantillas para generar y consultar tickets.

-Tests: Contiene archivos de pruebas utilizados para verificar la correcta funcionalidad del sistema.

-tickets.csv: Almacena los datos relacionados con los tickets generados.

-dataset1.csv: Contiene datos de los vuelos disponibles.

-passwords.txt: contraseñas de los usuarios (pilotos y sobrecargos).

-Archivos raíz del proyecto:

-requirements.txt: Lista las dependencias necesarias para ejecutar el proyecto.

-run.py: El archivo principal que inicia la aplicación.

-README.md: Contiene una breve descripción del proyecto y guías para la instalación.

-routes.py: Define las rutas que gestionan la navegación entre las diferentes vistas de la aplicación.

Esta estructura modular permite una fácil organización y expansión del proyecto, donde cada funcionalidad está separada en componentes lógicos, lo que facilita su comprensión y mantenimiento.

## **Dirigido para:**

Este proyecto está diseñado para ser utilizado por una variedad de usuarios que necesitan acceder a información sobre el clima en tiempo real. Los principales grupos de usuarios son:

### **Turistas:**

Los turistas que viajan a diferentes destinos pueden utilizar la aplicación para consultar el estado del clima en sus ciudades de destino. Esto les permite planificar mejor sus actividades y prepararse para las condiciones meteorológicas. La interfaz es sencilla y fácil de usar, diseñada específicamente para personas que no necesariamente tienen conocimientos técnicos, brindando la información de forma clara y directa.

### **Sobrecargos:**

El personal de vuelo, en particular los sobrecargos, necesita información meteorológica más detallada para garantizar la seguridad y comodidad de los pasajeros durante los vuelos. La aplicación proporciona datos adicionales, como la visibilidad y la humedad, que son relevantes para sus funciones. Los sobrecargos pueden acceder a estos datos mediante un sistema de autenticación.

### **Pilotos:**

Los pilotos requieren información meteorológica precisa y detallada, no solo sobre las condiciones en tierra, sino también sobre factores críticos como la velocidad y dirección del viento, la presión atmosférica y las coordenadas geográficas. Este tipo de usuario tiene acceso a datos avanzados que les permiten tomar decisiones informadas durante el vuelo. También cuentan con un sistema de autenticación para acceder a esta información.

Este sistema está diseñado para ser flexible y adaptarse a las necesidades específicas de cada tipo de usuario, proporcionando la información adecuada de forma clara y rápida, tanto para quienes buscan un uso básico como para quienes requieren detalles técnicos más avanzado

## **Roles**

Este proyecto fue desarrollado por un equipo de tres personas, cada una con un rol específico y tareas asignadas que contribuyeron al éxito del sistema.

Diego Hazael Vega Alonso - Desarrollador Backend y Testing:

Responsabilidades: Implementación de la entrada robusta para garantizar que los datos ingresados por el usuario sean validados y procesados correctamente. Manejo de los vuelos, integrando los datos y ofreciendo opciones de consulta precisas para el usuario. Desarrollo de tests para asegurar la correcta funcionalidad del sistema y verificar que el proyecto opere sin errores.

Alan Díaz Quijada - Desarrollador Backend y Arquitecto del Proyecto:

Responsabilidades: Diseño de la estructura del proyecto, organizando los módulos y la lógica interna del sistema. Manejo de la información del clima, integrando las consultas a servicios externos y gestionando los datos meteorológicos. Creación y manejo de tickets, desarrollando las funcionalidades para generar y consultar tickets, permitiendo a los usuarios acceder a la información del clima relacionada con sus vuelos.

Lizbeth Navarro - Desarrolladora Frontend:

Responsabilidades: Diseño e implementación de la interfaz de usuario (UI), asegurando que la experiencia de usuario (UX) sea clara y fácil de navegar. Creación de las vistas HTML y los estilos CSS personalizados para las páginas de consulta de clima y vuelos, así como la gestión dinámica de datos a través de plantillas



En dado caso, en que el proyecto sea aceptado, los roles en el sistema serían:

Usuarios en general:

Usuarios que acceden a la aplicación para consultar el clima o generar tickets. No necesitan permisos especiales.

Administrador de contenido:

Persona encargada de gestionar las plantillas HTML y los elementos visuales del sistema, asegurando que la información sea clara y actualizada.

Web Master:

Responsable del funcionamiento técnico completo del sistema, incluyendo servidores, bases de datos, y la resolución de problemas técnicos.

Desarrollador:

Persona encargada de realizar mejoras en el código y corregir errores en la aplicación una vez que ya está en producción.

## Información de Software

### *Tipo de software:*

Este proyecto consiste en una **aplicación web** que utiliza una arquitectura basada en el lenguaje de programación **Python** con el marco de trabajo **Flask** para el desarrollo de su servidor backend. La aplicación se complementa con **HTML** y **CSS** para la estructura y el diseño visual del frontend, lo que permite crear una interfaz de usuario amigable e interactiva.

Dicha aplicación está diseñada para ejecutarse en un entorno basado en **Linux**, lo que asegura compatibilidad con servidores Unix y sistemas de código abierto. El software puede clasificarse como un **proyecto de software web** de código abierto, que ofrece una combinación de tecnologías frontales y back-end para facilitar la creación de sitios web dinámicos o sistemas orientados a la web.

El uso de Flask como microframework permite desarrollar una aplicación ligera y modular, ideal para proyectos pequeños o medianos, en los que la velocidad y la escalabilidad son importantes. Al correr sobre Linux, la aplicación se beneficia de la robustez y seguridad del sistema operativo, garantizando un alto rendimiento y capacidad de gestión.

### ***Requerimientos:***

Los **requerimientos** para este proyecto están definidos en el archivo **requirements.txt**, el cual contiene las siguientes dependencias necesarias para el correcto funcionamiento de la aplicación:

**1.Flask==2.2.5:** Este microframework para Python es fundamental para el desarrollo del servidor web. Proporciona las herramientas básicas para manejar rutas, solicitudes HTTP y gestionar el ciclo de vida de la aplicación web.

**2.requests==2.31.0:** Una librería popular para realizar solicitudes HTTP, utilizada para interactuar con APIs externas o enviar datos entre diferentes partes de la aplicación de manera sencilla.

**3.pandas==2.2.2:** Librería utilizada para el manejo y análisis de datos, que permite trabajar con estructuras de datos como DataFrames, útiles en el procesamiento de información tabular o series temporales.

**4.levenshtein==0.12.0:** Esta biblioteca implementa el algoritmo de distancia de Levenshtein, utilizado para medir la similitud entre dos cadenas de texto. Es útil para realizar comparaciones de texto y detectar errores tipográficos o diferencias entre cadenas.

**5.pytest==8.3.3:** Un marco de pruebas utilizado para realizar testing automatizado en el proyecto, asegurando que el código sea fiable y se comporta como se espera bajo diferentes condiciones.

***Enfocado:***

Esta aplicación web está enfocada en proporcionar **información meteorológica en tiempo real** y **datos geográficos** de diversas ciudades a nivel mundial. Su principal objetivo es ofrecer un servicio práctico y útil en **aeropuertos**, siendo utilizada por **turistas, pilotos y sobrecargos**. La aplicación adapta la presentación de los datos de forma personalizada según el tipo de usuario, garantizando que cada grupo reciba información relevante para sus necesidades.

Por ejemplo, los **pilotos** pueden obtener detalles cruciales sobre las condiciones meteorológicas que afectan la seguridad del vuelo, mientras que los **turistas** reciben información más general sobre el clima, orientada a sus planes de viaje. Los **sobrecargos**, por su parte, obtendrán detalles que les ayuden en su logística y operación durante los vuelos.

La flexibilidad y precisión de los datos proporcionados hacen de esta aplicación una herramienta ideal para el entorno aeroportuario.

## Pseudocódigo:

### 1. Punto de entrada (`run.py`)

El archivo `run.py` es el punto de entrada principal de la aplicación. Aquí se define el proceso que inicializa y ejecuta el servidor de Flask.

#### Explicación:

- Se llama a la función `create_app()` (que se encuentra en `__init__.py`) para crear una instancia de la aplicación Flask.
- Las configuraciones de host, puerto y el modo de depuración (debug) se obtienen de las variables de entorno, o se utilizan valores predeterminados si no están configuradas.
- Finalmente, la aplicación se ejecuta utilizando el método `app.run()`, lo que inicia el servidor.

## Pseudocódigo:

```
INICIAR aplicacion = create_app() # Crear la aplicación Flask
```

```
SI __main__: # Verifica si el script se está ejecutando directamente
```

```
    OBTENER host = variable de entorno FLASK_RUN_HOST (por defecto '127.0.0.1')
```

```
    OBTENER port = variable de entorno FLASK_RUN_PORT (por defecto 5000)
```

```
    OBTENER debug_mode = variable de entorno FLASK_DEBUG (por defecto 'True')
```

```
    EJECUTAR aplicacion en host, port, con debug_mode # Iniciar la aplicación Flask
```

## 2. Inicialización de la aplicación (`__init__.py`)

Este archivo inicializa la aplicación Flask y define la estructura general.

### Explicación:

- `create_app()` es responsable de crear la instancia de la aplicación y configurar todo lo necesario.
- En este caso, se registra un **Blueprint** llamado `main`, que agrupa todas las rutas definidas en `routes.py`. Esto facilita la organización del código.

### Pseudocódigo:

FUNCION `create_app()`:

CREAR instancia de Flask # Se crea la aplicación Flask

REGISTRAR blueprint 'main' # Se registran las rutas de la aplicación en el blueprint

DEVOLVER aplicación Flask # Devuelve la instancia de Flask

### 3. Rutas de la aplicación (`routes.py`)

#### Ruta principal (`/`)

Esta ruta maneja la consulta del clima según la ciudad y el tipo de usuario.

#### Explicación:

- Al acceder a la ruta principal (`/`), la aplicación carga las contraseñas de un archivo de texto (`passwords.txt`).
- En el caso de una solicitud `POST` (cuando el usuario envía el formulario), se validan los datos ingresados (ciudad, tipo de usuario, contraseña).
- Si todo es válido, se llama a la función `get_weather_data_for_user()` para obtener el clima para la ciudad solicitada según el tipo de usuario (turista, sobrecargo, piloto) y se muestra en la plantilla `result.html`.

#### Pseudocódigo:

RUTA `/`, métodos = [`'GET'`, `'POST'`]

LLAMAR a `load_passwords()` PARA obtener contraseñas

SI contraseñas no existen:

MOSTRAR plantilla `'index.html'` CON mensaje de error

SI método es `POST`:

OBTENER ciudad, tipo\_usuario, contraseña desde el formulario

LLAMAR a `validate_form_data()` PARA validar los datos del formulario

SI hay error en los datos:

MOSTRAR plantilla `'index.html'` CON mensaje de error

LLAMAR a `get_weather_data_for_user()` PARA obtener los datos del clima

SI hay error al obtener clima:

MOSTRAR plantilla 'index.html' CON mensaje de error

MOSTRAR plantilla 'result.html' CON datos de clima

### **Ruta para generación de tickets (/ticket)**

Esta ruta permite al usuario generar un ticket basado en la ciudad de origen y destino.

#### **Explicación:**

- Cuando el usuario envía un formulario con las ciudades de origen y destino, se validan dichas entradas y se genera un ticket.
- El ticket se guarda en un archivo CSV, y luego se muestra el ticket generado en una plantilla HTML.

#### **Pseudocódigo:**

ruta '/ticket', métodos = ['GET', 'POST']

SI método es POST:

OBTENER origen y destino desde el formulario

LLAMAR a process\_ticket\_entries() PARA validar origen y destino

SI hay error en las entradas:

MOSTRAR plantilla 'ticket.html' CON mensaje de error

GENERAR ticket LLAMANDO a generate\_ticket()

GUARDAR ticket en CSV LLAMANDO a save\_ticket\_to\_csv()

MOSTRAR plantilla 'ticket\_result.html' CON el ticket generado

MOSTRAR plantilla 'ticket.html'

### **Ruta para consultar por ticket (/consultar\_por\_ticket)**

Esta ruta permite consultar el clima asociado a un ticket generado previamente.



**Explicación:**

- El usuario ingresa un ticket y sus credenciales. Luego, el ticket se busca en un archivo CSV para encontrar la ciudad de destino asociada.
- Si se encuentra, se consulta el clima para esa ciudad, y se muestra en la plantilla `result.html`.

**Pseudocódigo:**

RUTA '/consultar\_por\_ticket', métodos = ['GET', 'POST']

CARGAR contraseñas LLAMANDO a `load_passwords()`

SI contraseñas no existen:

MOSTRAR plantilla 'ticket\_search.html' CON mensaje de error

SI método es POST:

OBTENER ticket, tipo\_usuario, contraseña desde el formulario

LLAMAR a `validate_form_data()` PARA validar ticket, tipo de usuario y contraseña

BUSCAR ticket en CSV LLAMANDO a `search_ticket_in_csv()`

SI no se encuentra el ticket:

MOSTRAR plantilla 'ticket\_search.html' CON mensaje de error

OBTENER clima de la ciudad de destino LLAMANDO a `get_weather_data_for_user()`

SI hay error en el clima:

MOSTRAR plantilla 'ticket\_search.html' CON mensaje de error

MOSTRAR plantilla 'result.html' CON datos del clima, origen, destino

#### 4. Servicios de Clima (`weather_service.py`)

El archivo `weather_service.py` contiene la lógica para interactuar con la API de clima y formatear los datos según el tipo de usuario.

##### Explicación:

- La clase `WeatherService` carga una clave API desde un archivo y mantiene un sistema de caché para almacenar temporalmente los datos del clima.
- Dependiendo del tipo de usuario, los datos del clima se formatean de manera diferente. Esto se realiza en el método `create_weather_data()`.
- El servicio valida la ciudad, consulta la API y almacena en caché los resultados.

##### Pseudocódigo:

CLASE `WeatherService`:

FUNCION `__init__()`:

CARGAR `api_key` DESDE archivo '`api_key.txt`'

INICIALIZAR caché vacía

FUNCION `get_weather_data(ciudad, tipo_usuario)`:

VALIDAR ciudad LLAMANDO a `validate_city()`

SI hay error en la ciudad:

DEVOLVER error

SI los datos están en caché:

DEVOLVER datos desde caché

LLAMAR a `fetch_weather_data()` PARA obtener clima desde API

SI hay error con la API:

DEVOLVER error

CREAR datos del clima formateados según tipo de usuario LLAMANDO a `create_weather_data()`

ALMACENAR en caché

DEVOLVER datos del clima

FUNCION `create_weather_data(datos_clima, tipo_usuario)`:



SI tipo\_usuario es 'turista':

CREAR objeto TuristaWeatherData CON datos

SI tipo\_usuario es 'sobrecargo':

CREAR objeto SobrecargoWeatherData CON datos

SI tipo\_usuario es 'piloto':

CREAR objeto PilotoWeatherData CON datos

DEVOLVER datos formateados como diccionario

## 5. Validación y corrección de entradas (`Robust_Entry.py` y `city_helpers.py`)

Este módulo maneja la validación de las ciudades ingresadas por los usuarios.

### Explicación:

- Cuando un usuario ingresa una ciudad, `validate_city()` se encarga de corregir errores en el nombre y asegurarse de que es válida.
- Si la ciudad es inválida o necesita corrección, se devuelve un mensaje de error o sugerencia.

### Pseudocódigo:

FUNCION `validate_city(ciudad)`:

VALIDAR ciudad LLAMANDO a `RobustEntry()`

SI la ciudad tiene error:

DEVOLVER error

SI la ciudad tiene sugerencia:

DEVOLVER sugerencia

DEVOLVER ciudad corregida

## 6. Obtención de vuelos ([Available\\_Flights.py](#))

Este módulo recupera vuelos disponibles desde un archivo CSV según el origen o destino.

### Explicación:

- La clase `Flights` lee un archivo CSV que contiene un conjunto de vuelos.
- Según el origen o destino ingresado, filtra y devuelve los vuelos correspondientes.

### Pseudocódigo:

CLASE `Flights`:

FUNCION `__init__()`:

CARGAR dataset de vuelos desde 'dataset1.csv'

FUNCION `get_available_flights_by_destination(destino)`:

FILTRAR dataset de vuelos por destino

DEVOLVER lista de vuelos disponibles como diccionario

FUNCION `get_available_flights_by_origin(origen)`:

FILTRAR dataset de vuelos por origen

DEVOLVER lista de vuelos disponibles como diccionario

## 7. Generación y búsqueda de tickets (`ticket_helpers.py`)

Estas funciones permiten generar y buscar tickets en un archivo CSV.

### Explicación:

- `generate_ticket()` genera un ticket único basado en el origen y destino de un vuelo.
- Los tickets generados se guardan en un archivo CSV usando `save_ticket_to_csv()`.
- Se pueden buscar tickets en el CSV con `search_ticket_in_csv()`.

### Pseudocódigo:

FUNCION `generate_ticket(origen, destino)`:

    CONCATENAR origen + destino

    CREAR hash SHA256 del string concatenado

    EXTRAER primeros 6 caracteres del hash como ticket

    DEVOLVER ticket

FUNCION `save_ticket_to_csv(origen, destino, ticket)`:

    ABRIR archivo 'tickets.csv' en modo añadir

    GUARDAR origen, destino, y ticket en el archivo

FUNCION `search_ticket_in_csv(ticket)`:

    LEER archivo 'tickets.csv'

    BUSCAR ticket

    SI se encuentra:

        DEVOLVER origen y destino asociados al ticket

    DEVOLVER None si no se encuentra

## 8. Caché (`cache_helpers.py`)

Este módulo permite almacenar en caché los datos del clima por un periodo de tiempo para mejorar el rendimiento.

### Explicación:

- `is_cached()` verifica si los datos están en caché y si aún son válidos (no tienen más de 10 minutos).
- `store_in_cache()` guarda los datos del clima en la caché con una marca de tiempo.

### Pseudocódigo:

FUNCION `is_cached(caché, cache_key)`:

SI `cache_key` existe en caché y tiene menos de 10 minutos:  
DEVOLVER True  
DEVOLVER False

FUNCION `store_in_cache(caché, cache_key, datos)`:

ALMACENAR datos en caché con la hora actual

## 9. Formateo de datos meteorológicos (`data_formatter.py`)

Aquí se define cómo se formatean los datos del clima según el tipo de usuario.

### Explicación:

- Existen tres clases (`TuristaWeatherData`, `SobrecargoWeatherData`, `PilotoWeatherData`) que heredan de `WeatherData`. Cada una tiene su propio formato de salida.
- Dependiendo del tipo de usuario, se muestra más o menos información sobre el clima.

### Pseudocódigo:

CLASE `WeatherData`:

    FUNCION `__init__`(datos):

        ALMACENAR ciudad y temperatura

    FUNCION `to_dict`():

        DEVOLVER datos de ciudad y temperatura como diccionario

CLASE `TuristaWeatherData` HEREDA `WeatherData`:

    FUNCION `__init__`(datos):

        ALMACENAR descripción del clima

    FUNCION `to_dict`():

        DEVOLVER datos con descripción del clima

CLASE `SobrecargoWeatherData` HEREDA `WeatherData`:

    FUNCION `__init__`(datos):

        ALMACENAR humedad y visibilidad

    FUNCION `to_dict`():

        DEVOLVER datos con humedad y visibilidad

CLASE `PilotoWeatherData` HEREDA `WeatherData`:

    FUNCION `__init__`(datos):

        ALMACENAR latitud, longitud, velocidad del viento, dirección del viento y presión

    FUNCION `to_dict`():

        DEVOLVER datos con latitud, longitud, velocidad y dirección del viento, y presión



## 10. Validación de formularios (`form_helpers.py`)

Este módulo se encarga de extraer y validar los datos enviados por los usuarios a través de los formularios en el frontend.

### Explicación:

- `get_form_data()` y `get_ticket_form_data()` extraen los datos del formulario enviado.
- `validate_form_data()` se asegura de que los datos ingresados sean correctos, validando la ciudad, tipo de usuario y contraseña.

### Pseudocódigo:

FUNCION `get_form_data()`:

EXTRAER ciudad, tipo\_usuario, contraseña desde el formulario  
DEVOLVER valores

FUNCION `validate_form_data(ciudad, tipo_usuario, contraseña, USER_PASSWORDS)`:

SI ciudad o tipo\_usuario no están presentes:  
DEVOLVER mensaje de error


SI tipo\_usuario es 'sobrecargo' o 'piloto':  
VALIDAR contraseña CON USER\_PASSWORDS  
SI la contraseña es incorrecta:  
DEVOLVER mensaje de error

DEVOLVER None SI no hay errores

## Página principal y secundarias

Nos enfocamos en implementar un diseño que sea legible y fácil de usar.

### Página principal:



CLIMA EN TIEMPO REAL

CONSULTA EL CLIMA POR CIUDAD

**Ingrese los datos**

Ciudad:

Seleccionar tipo usuario:

Turista

Consultar clima

¿QUIERES CONSULTAR EL CLIMA CON UN TICKET? [DA CLICK AQUI EN CONSULTAR CON TICKET](#)

¿NO CONOCES EL TICKET DE TU VUELO? [GENERAR TICKET](#)

### Página principal al solicitar contraseña:



CLIMA EN TIEMPO REAL

CONSULTA EL CLIMA POR CIUDAD

**Ingrese los datos**

Ciudad:

Seleccionar tipo usuario:

Sobrecargo

Contraseña:

Ingrese su contraseña

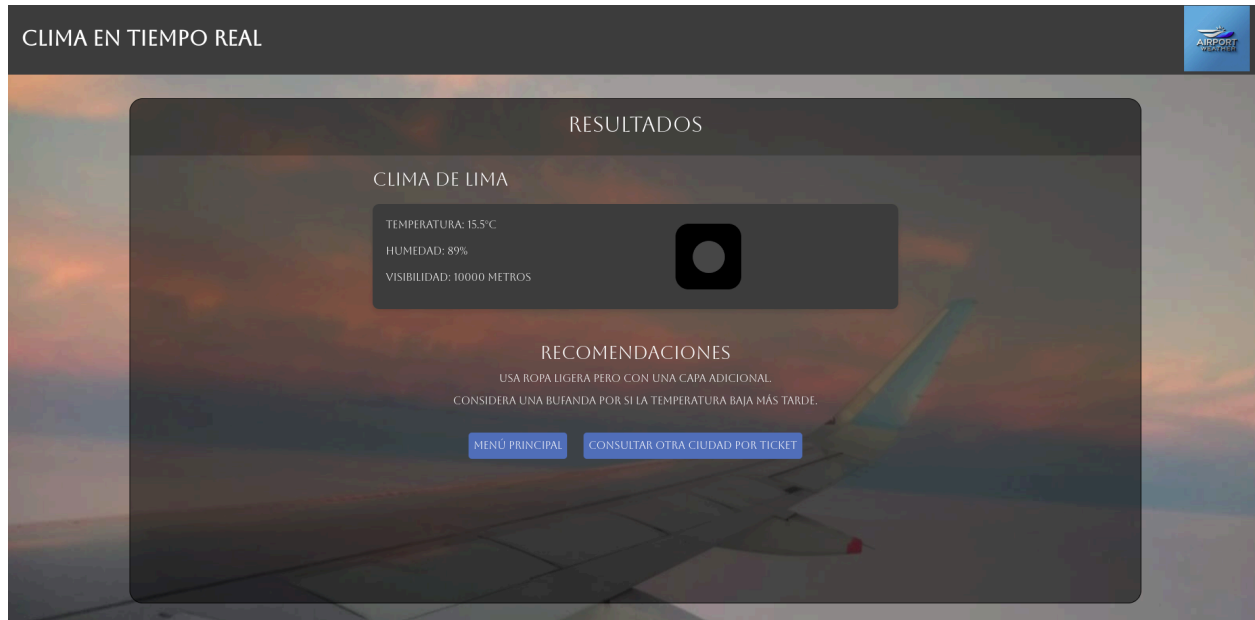
Consultar clima

¿QUIERES CONSULTAR EL CLIMA CON UN TICKET? [DA CLICK AQUI EN CONSULTAR CON TICKET](#)

¿NO CONOCES EL TICKET DE TU VUELO? [GENERAR TICKET](#)

## Página secundarias:

### 1)Página de resultados



CLIMA EN TIEMPO REAL

RESULTADOS

CLIMA DE LIMA

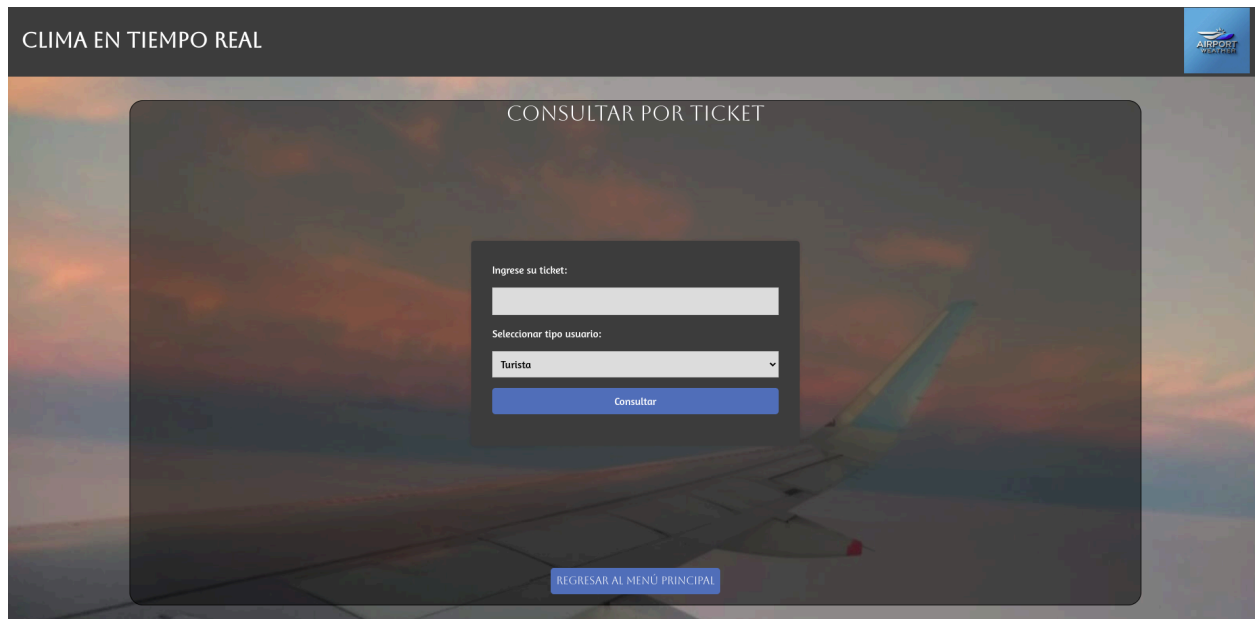
TEMPERATURA: 15.5°C  
HUMEDAD: 89%  
VISIBILIDAD: 10000 METROS

RECOMENDACIONES

USA ROPA LIGERA PERO CON UNA CAPA ADICIONAL.  
CONSIDERA UNA BUFANDA POR SI LA TEMPERATURA BAJA MÁS TARDE.

MENU PRINCIPAL CONSULTAR OTRA CIUDAD POR TICKET

### 2)Consultar con ticket



CLIMA EN TIEMPO REAL

CONSULTAR POR TICKET

Ingrese su ticket:

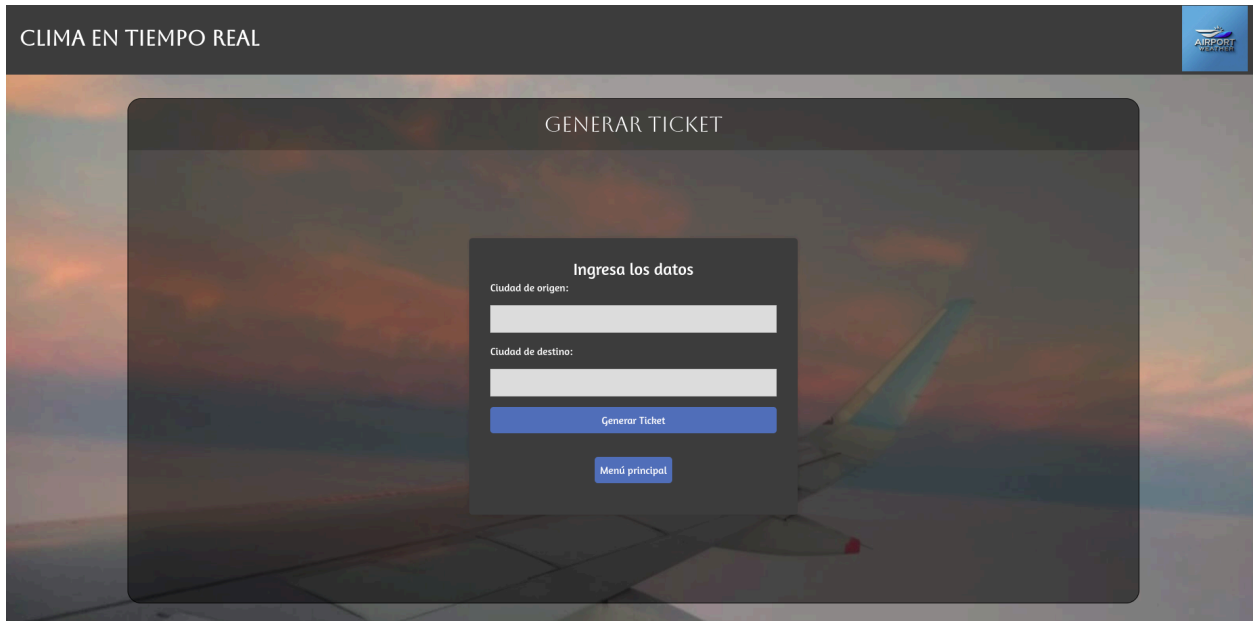
Seleccionar tipo usuario:

Turista

Consultar

REGRESAR AL MENU PRINCIPAL

### 3) Generar un ticket

A screenshot of a web application interface for generating a ticket. The background is a dark, atmospheric image of an airport tarmac at dusk or dawn. A dark grey rectangular box is centered on the screen, containing a form titled "Ingresa los datos". The form has two input fields: "Ciudad de origen:" and "Ciudad de destino:". Below these fields are two buttons: a blue "Generar Ticket" button and a smaller, lighter blue "Menú principal" button. The top of the page has a dark header with the text "CLIMA EN TIEMPO REAL" on the left and a small "AIRPORT WEATHER" logo on the right.

### Imágenes y partes gráficas

Para este proyecto decidimos ocupar imágenes de nuestra propiedad, es decir nosotros somos los autores de estas fotografías/imágenes, fueron seleccionadas de tal manera que dieran una buena visualización a la app.

Imagen para el logo:



## **Colores y tipografías**

Gris oscuro: El encabezado superior y el área alrededor del formulario tienen un fondo gris oscuro, lo que le da un aspecto sobrio.

Blanco: El texto "CLIMA EN TIEMPO REAL" y "GENERAR TICKET" está en blanco, destacando sobre el fondo oscuro.

Negro translúcido: El fondo del cuadro de entrada (el formulario) tiene un color negro translúcido, lo que permite ver ligeramente el fondo de la imagen detrás.

Azul: Los botones de "Generar Ticket" y "Menú principal" son de un tono de azul intenso, lo que les da un aspecto resalta sobre el formulario oscuro.

Imagen de fondo: En el fondo, predomina una imagen con tonos cálidos, donde destacan colores como el rosa anaranjado del atardecer, junto con tonos de azul y gris en el cielo.

## **Procedimientos**

### **Mantenimiento**

#### **1.Verificación de la Integración de la API**

Revisión periódica de la conexión con OpenWeather: Asegurar de que la API sigue proporcionando datos actualizados y correctos. Verificar si han habido cambios en el endpoint o en los parámetros requeridos. Actualizar las claves de API si es necesario.

Monitoreo de la tasa de uso de la API: Verificar que el límite de solicitudes no se esté excediendo y ajustar la lógica para evitar bloqueos o errores en caso de que se llegue al límite.

Manejo de imágenes proporcionadas por la API: Asegurar que las imágenes relacionadas con el clima se estén mostrando correctamente y que su formato no haya cambiado.

## 2. Gestión de Roles y Autenticación

Prueba periódica del sistema de autenticación: Verificar que los roles de piloto y sobrecargo puedan acceder a los resultados sin problemas, mediante la correcta verificación de la contraseña.

Fortalecimiento de seguridad en contraseñas: Revisa el almacenamiento de contraseñas.

## 3. Validación de Entradas del Usuario

Validación de la entrada de la ciudad: Que el sistema maneje correctamente los errores cuando los usuarios ingresan mal el nombre de la ciudad.

Revisar que los mensajes de error sean claros y útiles tanto para la autenticación como para la entrada de la ciudad.

## 4. Optimización de la Interfaz y Diseño

Revisión del diseño de la página de resultados: Mantener la interfaz actualizada, asegurando que los datos del clima y la imagen relacionada se visualicen correctamente. 5. Optimización del Rendimiento

## 7. Actualización del Contenido

Revisar y actualizar el contenido periódicamente para que sea relevante (de las recomendaciones).

## 8. El costo de este mantenimiento:

Mantenimiento preventivo o menores ajustes: Tarifa mensual fija, entre \$1,550 y \$4,300.

Mantenimiento correctivo o mejoras mayores: Por horas trabajadas, entre \$500 y \$960, dependiendo de la magnitud de las tareas.

## **Futuras Actualizaciones**

**Botones y formularios:** Actualmente los botones de "Generar Ticket" y "Menú principal" son funcionales, pero podrían mejorarse visualmente añadiendo íconos relacionados (un icono de avión, por ejemplo) y ajustando su tamaño para mejorar la experiencia de usuario.

**Separación de roles:** Crear una página de inicio que pregunte al usuario qué rol va a usar (Turista, Piloto, Sobrecargo) con iconos representativos para cada uno, ofreciendo una interfaz única dependiendo del rol seleccionado.

**Autenticación clara:** Para los roles de **piloto y sobrecargo**, mejorar el diseño de la interfaz de inicio de sesión, con campos de contraseña más estilizados y mensajes claros en caso de errores (contraseña incorrecta o ciudad no encontrada). Una animación simple de acceso, como una transición de la pantalla, podría mejorar la experiencia.

**Sistema de Tickets:** Mejorar el diseño del ticket de resultados para que sea más claro y atractivo

## **Elementos Visuales y Diseño Responsivo**

**Fondos dinámicos:** Cambiar el fondo de la página de acuerdo con el clima de la ciudad consultada, por ejemplo, cielos despejados si el clima es soleado, o un fondo nublado para climas lluviosos.

Cuando el usuario inicie sesión, usar una breve animación de carga que muestre un avión en movimiento o despegando.