**FLYDRONE DOCUMENTACIÓN**

**Diseño**

Desde un primer momento se ha pretendido hacer un diseño modular separando todas las funcionalidades de la web (js) en diferentes scripts en función a su librería. Además, se ha pretendido hacer una especia de Single Page APP mediante la introducción de objetos del DOM de forma dinámica. Como desde un primer momento no planteé bien la aplicación ni la idea fundamental de esta, he tenido que lidiar con algún que otro problema de dinamismo que ha hecho que la estructura modular del proyecto no haya quedado como la diseñe en mi cabeza.

El diseño de la aplicación lo he dividido en 2 partes principales:

-Primero la sección de **Login** que involucra un index.html estático junto con un script que usa la librería de Rainyday.js (la cual me ha dado muchos problemas) junto con su css y el script de la API de Firebase la cual permite al usuario logearse, registrarse y recuperar su contraseña.

-Una vez que el usuario se logea aparece la segunda sección, el **panel de usuario**. Cuando se logea el usuario se oculta toda la sección de registro y se muestra una nueva sección con un panel con toda la funcionalidad. Este panel ha sido creado de forma dinámica y podemos encontrar 4 botones y el logo de la app. Estos botones permiten verificar la viabilidad de vuelo (función que llama a las APIS de OpenStreetMap, Leaflet.js, dataTables.js y openMeteo.js), iniciar y detener la detección de obstáculos (que llaman a la API de TensorFlow.js y a uno de sus modelos preentrenados) y por último cerrar sesión.

**Flujo de ejecución**

1. El usuario entre a la se registra he inicia sesión.
2. La sección de registro se oculta, se elimina el efecto de la librería rainyday.js y se crear el panel de usuario de forma dinámica.
3. El usuario interacciona con la funcionalidad de la aplicación.
4. El usuario cierra sesión y se refresca la página devolviendo este al login.

**Dificultades encontradas**

Pese a ser una tarea destinada a la documentación y manejo de APIS considero que gracias a la información que he podido obtener de internet esto no ha sido lo más complejo del proyecto. Las mayores dificultades que me he encontrado a la hora de realizar el trabajo han sido:

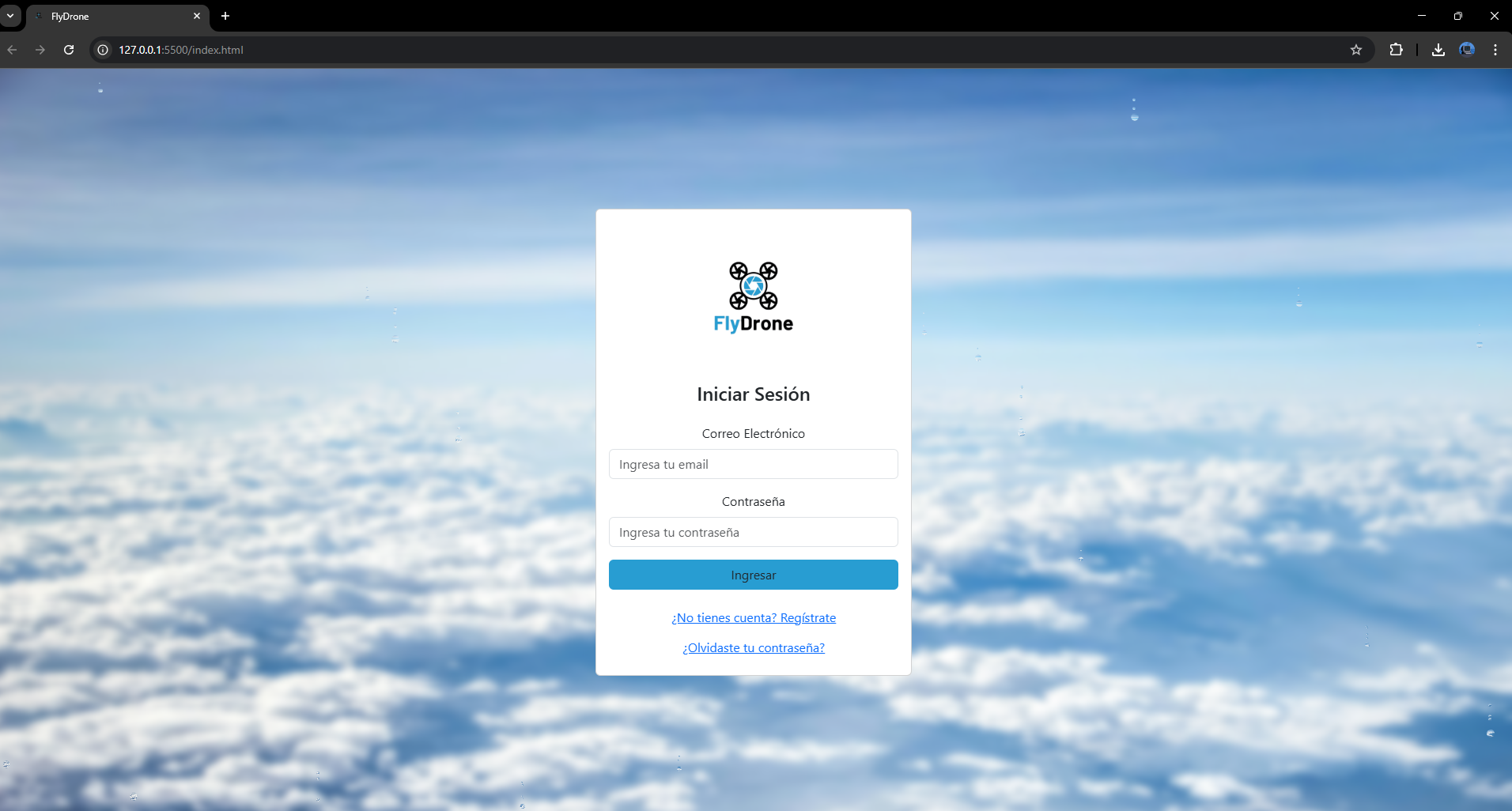
1. Uso de la librería Rainyday.js y compatibilidad. Aunque considero que crea un buen efecto visual y que combina con la temática de la aplicación es una librería que no volvería a utilizar dado que consume muchos recursos y aunque tiene métodos para paralizarla completamente como clear() que limpia las gotas de agua o stop() a mí no me han funcionado correctamente . Es más, a veces hasta paraliza la app. La he dejado por lo visual, pero en otro tipo de proyecto la hubiese eliminado.
2. SPA. Considero que desde un principio no he seguido la estructura más modular y dinámica y eso ha hecho que tenga muchos problemas de llamar a objetos del DOM cuando todavía no estaban ni pintando. Buscando en internet he podido encontrar una solución que es delegando el evento en el body y propagándolo, pero no creo que sea lo más eficiente.
3. He tenido que invertir bastante tiempo a la API de TensorFlow.js hasta encontrar un modelo adecuado.
4. He tenido muchos errores al implementar el hosting y la subida a Github. No entiendo del todo este error. He usado en todo el proyecto rutas relativas para evitar este tipo de erros. En local el proyecto me funciona correctamente pero al subirlo a firebase o githupages no accedía a algunos recursos. Lo he solucionado peleándome con ello y cambiando innumerables veces las rutas hasta que ha funcionado.

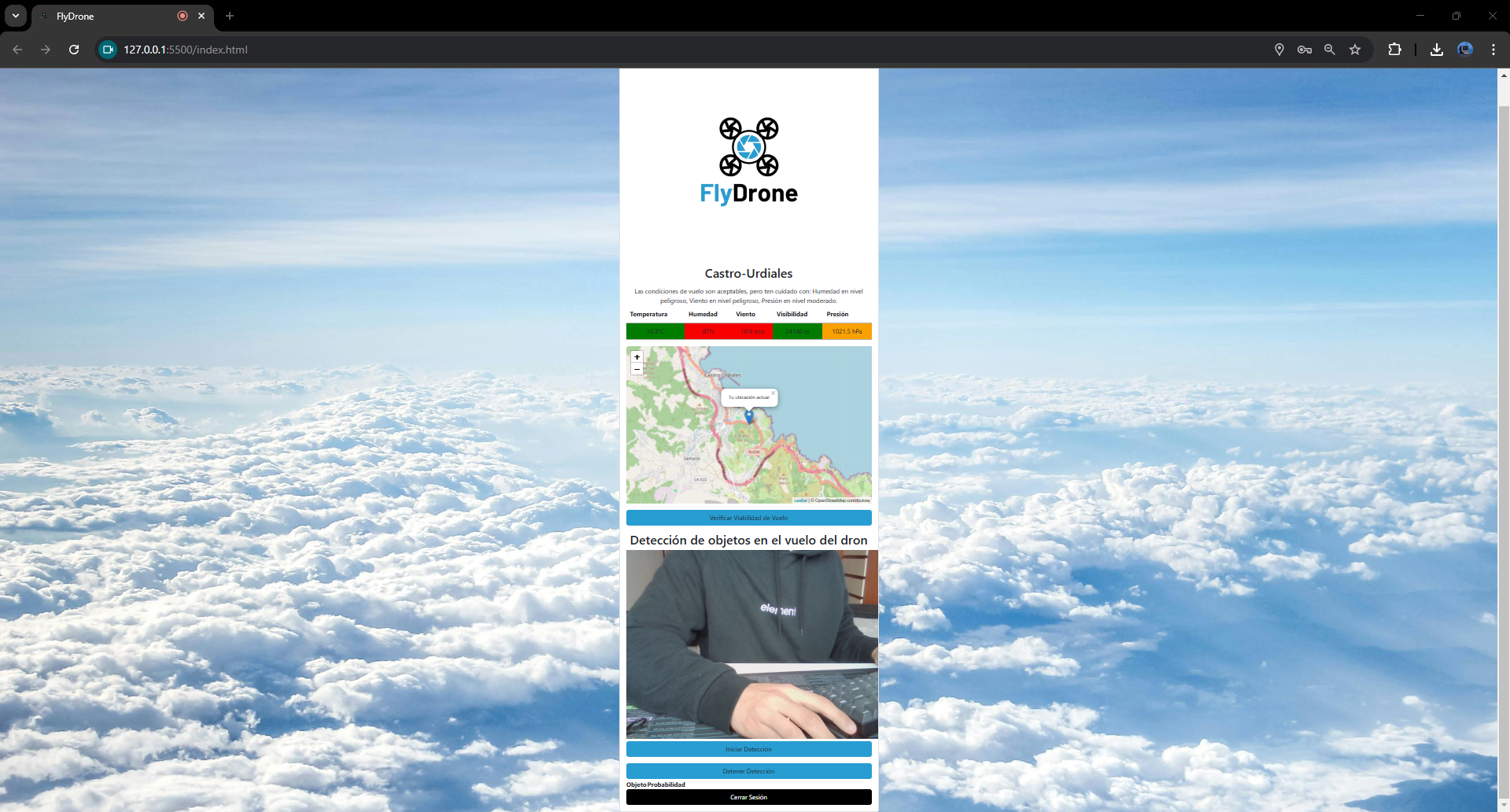
**Resumen**

Gracias a la ejecución asíncrona, la aplicación puede manejar datos y funciones sin bloquear la interfaz, permitiendo procesos en segundo plano sin afectar el rendimiento. Esto optimiza la carga de información en tiempo real y mejora la interacción del usuario, garantizando una navegación fluida.

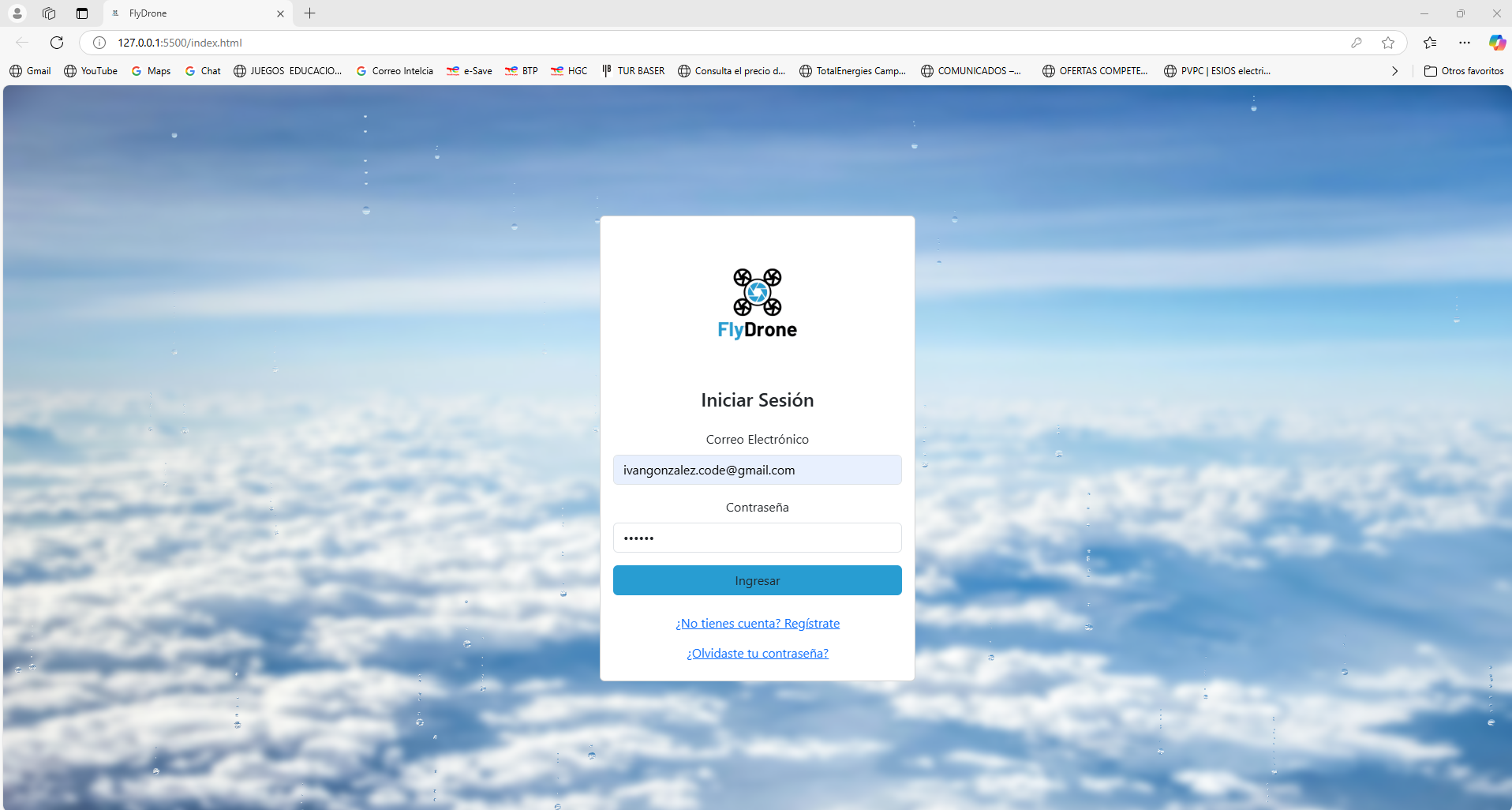
**Soporte Navegadores**

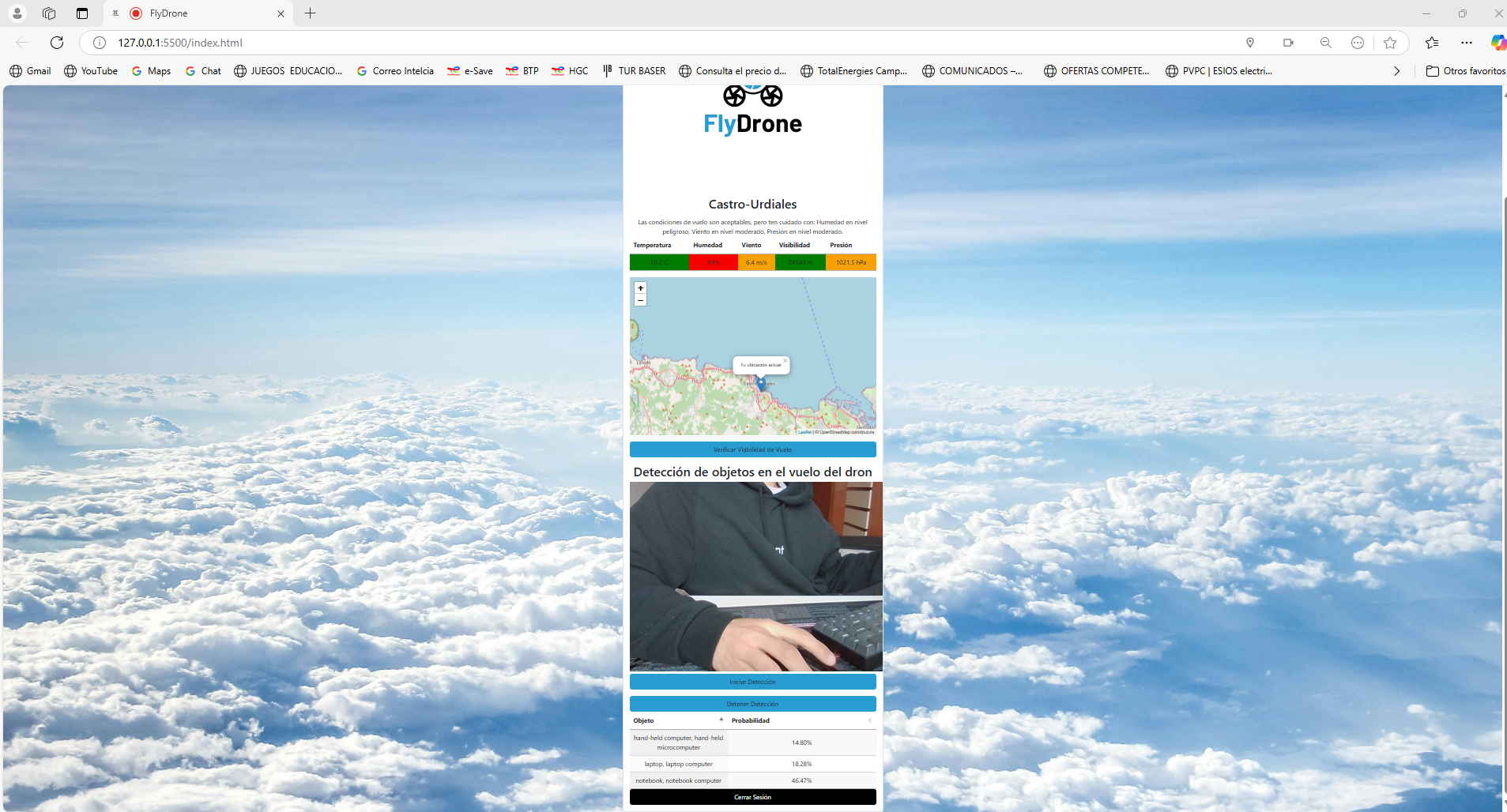
**Google**

****

****

**Edge**

****

****