

JURASSIC PARK SENSOR CONTROLLER

Se construyo un sistema de gestión de hábitats para dinosaurios. El sistema monitorea a tiempo real distintos tipos de sensores (a los que le llegan información simulada constantemente). Se utilizando una arquitectura reactiva y asíncrona para manejar múltiples flujos de datos concurrentes sin bloquear el sistema.

El backend del sistema a sido construido Python, utilizando para el funcionamiento del sistema FastAPI, RxPY, asyncio y Pydantic. Y un frontend con HTML, CSS y JavaScript.

Ademas se tiene un sistema de logs y métricas que registran, los movimientos en el sistema.

Arquitectura

-Models:

Se definió la estructura de los datos usando Pydantic para asegurar que ningún dato corrupto entre al sistema.

>Sensors.py

Aquí es donde definimos la base de los sensores que hay en el sistema, tenemos la clase SensorBase que indica las bases que tendrán cada tipo.

Se crearon 3 tipos de sensores, cada uno con sus propios parámetros que definen acciones que pasaran en el sistema. Movimiento (MotionSensor), temperatura (TemperatureSensor) y pulsaciones (HeartRateSensor).

>Infrastructure.py

Este archivo define las entidades físicas del parque. Se tiene una clase que define hábitats de dinosaurios, con un tamaño definido para que el sensor de movimiento pueda detectar por áreas y si por ejemplo alguno de los dinosaurios se sale de los límites. Aparte también se le define una temperatura, que es monitorizada por su respectivo sensor. Y tiene una lista donde se guardarán los dinosaurios que están en ese hábitat.

Luego tenemos la clase que define el parque en el que guardaremos los diferentes hábitats. Se pueden tanto añadir como eliminar por el usuario.

>dinosaur.py

Define el objeto de dinosaurio con sus características únicas. Este se podrá meter luego en la lista que tiene hábitats.

>alert.py

Clase base de la alerta que le asigna una id la hora a la que surge, de que sensor, el nivel de alarma y una descripción del suceso.

-Services

Es donde se encuentra la lógica del sistema y el procesamiento reactivo.

>evaluator.py

Este es el archive que se encarga de la lógica tras las alertas de los sensores, evalúa si los datos que entran cambian de los que deberían.

En caso de la temperatura (evaluate_temperature), entra la lectura de la temperatura y el objeto hábitat y verifica si la temperatura actual se desvía más de 5 grados de la mean_temperature ideal del hábitat. Devuelve alerta en caso de ser mucho frío o calor.

Caso de movimiento (evaluate_motion), lee movimiento y hábitat. Revisa la intensidad (> 8 es sospechoso) y revisa la altura si el movimiento ocurre a una altura (z) mayor que la altura de la valla del hábitat (habitat.size.z), confirma una fuga.

Y para el ritmo cardiaco (evaluate_heart_rate), hace una lectura cardiaca del dinosaurio y compara los BPM actuales con el ritmo base del dinosaurio. Si supera el 50% extra (ej. base 100, actual 150), dispara alerta de estrés.

>simulator.py

Es el encargado de simular los datos constantes que procesa el sistema. Este tiene la lista de dinosaurios y hábitats para simular que de ellos vienen diferentes avisos.

Las funciones de create_temperature_stream, create_motion_stream, create_heart_rate_stream, se encargan de llamar cada x segundos de llamar a las funciones generadoras.

_generate_random_temp, _generate_random_motion, _generate_random_bpm, se encargan de elegir un hábitat o dinosaurio al azar y genera valores que a veces pueden ser alertas.

>stream_manager.py

Se encarga de mover los datos de un lado a otro. Se define una tubería y un buffer.

|On_sensor_data, recibe datos del simulador y los mete en el flujo.

|_process_batch, se ejecuta cada vez que el Buffer se llena. Recorre la lista de eventos y se los pasa uno a uno a _analyze_reading.

|_analyze_reading, busca el objeto Hábitat o Dinosaurio correspondiente al ID del sensor. Llama al RuleEvaluator correspondiente. Si el evaluador devuelve Alerta: Incrementa el contador de alertas y escribe el mensaje ALERT... en el log (que luego lee el frontend).

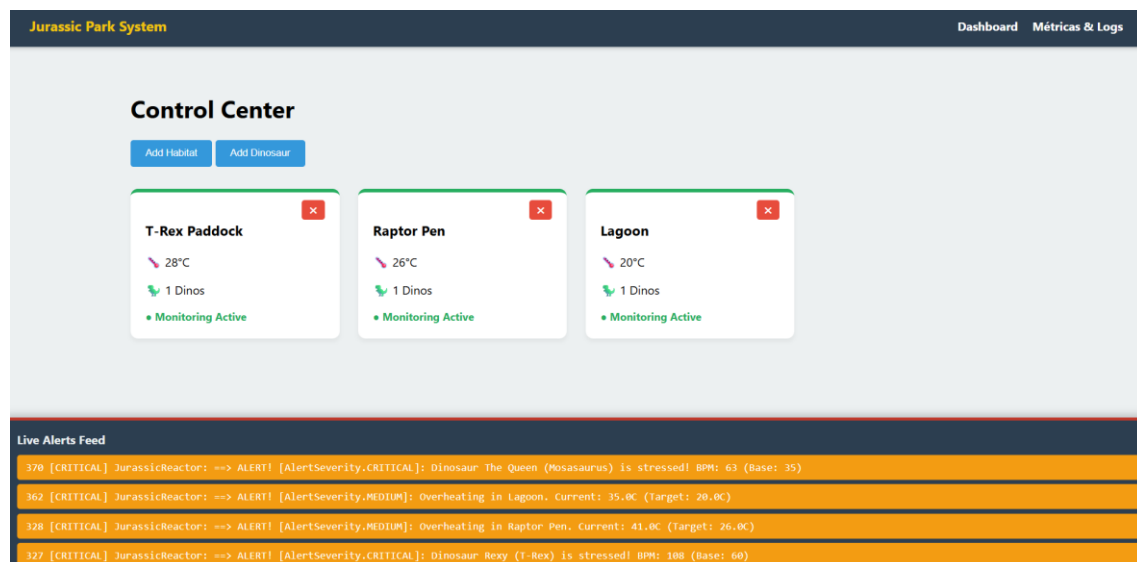
API

Se optó por una arquitectura Server-Side Rendering (SSR) híbrida. FastAPI sirve el HTML base, pero JavaScript actualiza los datos dinámicamente.

El frontend se divide en 3 páginas:

-Dashboard:

Es la pagina principal en el que se muestran los hábitats. Se puede agregar o eliminar tanto hábitats como dinosaurios, además abajo aparecen las alertas que puedan surgir de forma dinámica.



-Información de hábitats:

Al pinchar desde la pagina principal en cualquiera de los hábitats te saldrá una página con información más específica sobre esta.

← Back to Dashboard

T-Rex Paddock

ID: aaf41bb5-abf7-4227-8280-8b1bf4d9706f

Environment

Temperature Target: 28.0°C

Current Status: Active


Dimensions (Meters)

• Width (X): 800.0m

• Length (Y): 800.0m

• Height (Z): 25.0m

Inhabitants

 Dino ID:337970f0-48a0-4cd8-a69a-93f5a10018d2

-Métricas:

La pagina que mide los datos recopilados sobre el funcionamiento del programa.

