Airguardian

Backend - Mini Version

info@hive.fi

Contents

Chapter 1

INTRODUCTION

Mission Briefing

You are part of an elite airspace surveillance team tasked with **protecting critical infrastructure from unauthorized drone activity**. Drones are becoming increasingly common, and while many serve legitimate purposes, some stray into restricted airspace—whether by accident or with **malicious intent**. Your mission is to develop a **real-time drone monitoring system to detect intrusions and flag violations**.

Imagine this: A drone is approaching a **military radar station**, flying too close to a restricted zone. If it enters the **1,000-unit No-Fly Zone** [**NFZ**] around the radar, it could interfere with sensitive operations or even gather intelligence. Your system will **track drone movements and detect violations**, ensuring that unauthorized drones don't go unnoticed.

This is a **simplified version** designed to test our API endpoints and can be completed in 2-3 days.

Chapter 2

GENERAL INSTRUCTIONS

2.1 What is expected from you?

- You will use **GitHub** to push and manage your project code.
- You must provide a **detailed README.md** file that explains at least:
  - What the project does
  - How to install dependencies
  - How to run the application locally
  - Any relevant architectural or usage notes
- **Do not hardcode secrets or external API URLs** in your codebase.
  - Use a **.env** file to store environment variables.
  - Include a **.env.example** file in the root of the repository to document expected variables.
- Use **Poetry** for managing and locking dependencies.
- You will build a **FastAPI**-based backend system that performs the following tasks:
  - **Collects drone position data** [x, y, z] from the external API at scheduled intervals.
  - **Detects violations** when a drone enters the 1,000-unit NFZ radius, centered at [0, 0].
  - **Fetches drone owner information** from the other endpoint, but only for drones that have violated the NFZ.
  - **Stores all detected violations** in a PostgreSQL database under a table named violations.
  - **Exposes API endpoints** to retrieve real-time drone data and a list of violations that occurred within the last 24 hours.

This is a simplified version designed for testing our API endpoints and can be completed in 2-3 days. Focus on core functionality rather than advanced features.

2.2 What will you learn?

By completing this phase, you will:

- Gain hands-on experience with **FastAPI** and **Pydantic** for API development and data validation.
- Learn how to **schedule background tasks** using **Celery**.
- Work with **databases** to store and retrieve structured violation data.
- Implement **geospatial calculations** to detect drones within the NFZ.
- Handle **basic error management and logging** for a backend system.

Chapter 3

MANDATORY PART

To complete this project, you must implement the following core features.

3.1 Fetching Drone Position Data

Your application must periodically collect drone positions from the external source at:
https://drones-api.hive.fi/drones/

Specifically, it should:

- Fetch the x, y, and z coordinates of each drone, along with its **owner_id**.
- Use **Celery** to schedule this task at a fixed interval of 10 seconds.
- Ensure that drone data is processed reliably even if temporary network failures occur.

Note: Drone positions are returned within a bounded 3D coordinate space, where:

- x and y range from **-10,000** to **10,000**
- z (altitude) ranges from **20** to **300**

While the z value is collected and stored, it is not used for NFZ violation detection (which is based solely on x and y).

Implement basic error handling around the data collection task to ensure reliability.

3.2 No-Fly Zone Violation Detection

The No-Fly Zone (NFZ) is defined as a circle with a radius of 1,000 units, centered at (0, 0). (One unit can be considered equivalent to

1 meter.)
A drone is considered to be **in violation** if its position falls **within or on the edge of this circle.**
Any drone **outside the radius** is considered safe.

When a violation is detected, the following data **must be logged:**

- **Drone ID**
- **Timestamp** (when the violation occurred)
- **Drone position:** x, y, z
- **Owner First Name** (retrieved from external API)
- **Owner Last Name** (retrieved from external API)
- **Owner Social Security Number** (retrieved from external API)
- **Owner Phone Number** (retrieved from external API)

You can retrieve the drone owner's personal information via:
https://drones-api.hive.fi/users/:owner_id

For example:
https://drones-api.hive.fi/users/add6ad16-c284-4304-a0e4-1cc39052adc3

3.3 Storing & Retrieving Violations

All detected violations **must be persisted** in a PostgreSQL database in a table named violations. This table should store all relevant information for each violation.

⚠ **Owner details must be fetched from the external API only when a violation is detected, to avoid unnecessary external calls.**

3.4 API Endpoints

All of your API endpoints **must be defined using Pydantic models**-both for **request parameters** and **response schemas**.

This ensures that:

- Incoming data is **validated** and structured.
- Outgoing responses are **consistent** and **predictable**.

3.4.1 /health

A basic health check endpoint used to confirm that the backend service is running and responsive.

When called, this endpoint should respond with an **HTTP 200 OK** status code and a simple JSON payload indicating success: **{"success": "ok"}**

3.4.2 /drones

This endpoint acts as a **proxy** to the external drone tracking API. Its main purpose is to **abstract the external service** (https://drones-api.hive.fi/drones) and **prevent direct exposure of the upstream URL** to clients.

When called, the backend will:

- Make a **GET** request to **https://drones-api.hive.fi/drones**
- Receive the current position data of all drones
- Return the data **as-is or with minimal processing** to the client

Note: Since this endpoint acts as a gateway to an external system, you should ensure basic **error handling and logging** to make the proxy reliable.

3.4.3 /nfz

This endpoint returns all drone violations detected within the **last 24 hours**.

No query parameters are required. The endpoint will always return violations from the last 24 hours.

Example:

- GET /nfz: Returns violations from the last 24 hours

Authentication Requirement:

Since this endpoint is exposing personal data it requires to be a minimum **protected**. It will only respond if the request includes a valid **X-Secret** header.

- The header must be present and match a pre-configured secret stored in your **.env** file.
- If the header is missing or incorrect, the API should return a **401 Unauthorized** response.

3.4.4 Error Handling & Logging

- Implement **basic error handling** for:
  - API failures: use standard HTTP error codes where appropriate.
  - Network timeouts
  - Invalid data formats
- Ensure **basic logging** is in place to track system activity.

3.5 What is the expected output?

At the end of this phase, you will have a functional backend service that:

- Fetches **real-time drone position data**.
- Detects **violations when drones enter the NFZ.**

- Stores violations **with timestamps and drone identifiers**.
- Provides API endpoints for **data retrieval and analysis**.
- Implements **basic error handling and logging.**

Chapter 4

FINAL WORDS

Your backend is the **foundation of Airspace Guardian.** A secure and efficient system will **ensure real-time detection of unauthorized drones**, helping protect restricted airspace. This mini version focuses on core functionality and can be completed in 2-3 days, making it perfect for testing our API endpoints.

Good luck, and may your code keep the skies safe!