**Guide for changes I make in forest fire detection project**

**Guide to Models in core/models.py**

**Overview**

This guide provides details on the Event and EventCount models implemented, focusing on the logic for:

1. Tracking event counts (total, day/night, species-based) for cameras.
2. Fetching and storing weather data using external APIs.
3. Handling updates when events are created or updated.

**1. Models**

I added these Models in models.py

**EventCount Model**

* Tracks the counts of various types of events for each camera.
* Fields:
  + camera: ForeignKey to the Camera model.
  + total\_event\_count: Total events recorded.
  + total\_night\_event\_count: Events during night hours.
  + total\_day\_event\_count: Events during day hours.
  + fire\_day\_event\_count: Fire-related events during the day.
  + smoke\_night\_event\_count: Smoke-related events during the night.
  + fire\_night\_event\_count: Fire-related events during the night.
  + smoke\_day\_event\_count: Smoke-related events during the day.
  + night\_event\_with\_more\_than\_one\_species: Night events with multiple species.
  + day\_event\_with\_more\_than\_one\_species: Day events with multiple species.

**Event Model**

Adding these fields into the Event table

1. **weather\_data**:

* **Type**: JSON Field
* **Purpose**: To store weather-related data such as temperature, humidity, or any other meteorological parameters relevant to the event.
* **Advantages**: Can store structured data directly. Easy integration with weather APIs.

1. **nasa\_tag**:

* **Type**: BooleanField
* **Default**: False
* **Purpose**: To indicate whether an event is tagged with some specific condition or criteria, such as NASA-related classifications or data.

1. **weather\_station**:

* **Type**: JSONField
* **Purpose**: To store detailed weather station data, potentially including fields like location, station ID, and timestamped records.
* **Advantages**:Keeps all weather station-related information together. Facilitates easy querying and updates.

**2. Key Features and Logic**

**A. Event Count Management**

* Implemented in the update\_event\_counts method of the Event model.
* Logic:
  + Fetch or create an EventCount instance for the associated camera.
  + Determine whether the event occurred during the day or night.
  + Count events with multiple species.
  + Increment specific counters based on species (fire, smoke) and time (day, night).

**Night vs. Day Logic:**

* Night: Hours >= 18 (6 PM) or <= 6 AM.
* Day: Hours > 6 AM and < 6 PM.

**Species-Based Counts:**

* Uses the species field (Many-to-Many) to check for specific species (fire, smoke) in the event.

**B. Weather Data Fetching:**

* **check\_weather\_station\_api**:
  + Fetches air temperature and humidity from a weather station API based on the camera's ID.
  + Adds the data to the weather\_station field.
* **get\_weather\_data**:
  + Fetches historical weather data from OpenWeather API based on the event's timestamp and camera's location.

**C. API Rate-Limiting**

* **check\_weather\_api\_rate\_limit**:
  + Limits API calls to:
    - 900 daily requests.
    - 55 requests per minute.
  + Uses Django's cache for rate-limiting logic.

**3. Usage**

**Event Creation Workflow**

1. Create or update an Event instance.
2. The save method:
   * Fetches weather data if not already available.
   * Updates the relevant EventCount instance.

**EventCount Updates**

* Total counts and species-specific counts are incremented based on event details.

**Weather Data**

* Automatically fetched during the save process.

**Guide to Api\_view in core/api/api\_views.py**

**WeatherDataAPIView**

**Key Features:**

* Filters data by camera\_id, start\_time, and end\_time.
* Converts Unix timestamps (in milliseconds) to datetime.
* Uses MQTT for PTZ control operations.

**EventCountViewSet**

**Key Features:**

* Filters events based on camera\_id.
* Deletes events and updates EventCount.
* Includes a destroy method for instance-level deletions.

**Guide to Command in core/management/commands**

I added these command files

1. **download\_kmz.py:** This command automates downloading and saving KMZ files from NASA's FIRMS API, making the files available for further processing or analysis in the Django project. It ensures reliability with error handling and logs for monitoring execution.
2. **export\_events\_to\_csv.py:** This command ensures events from the last two months are documented. Each camera is represented in the export, even if it has no recent events.
3. **fetch\_weather\_data.py:** This command automates the retrieval and storage of weather data for cameras form weather station install on camera location, enabling centralized and structured weather data storage for further analysis or use in applications. Constructs the API URL using the device\_id, telemetry keys (Air\_Temperature and Air\_Humidity), and timestamp range.
   1. **Token Expiry Information**
      1. The current authentication token is valid until 2025-03-16 01:36:52.
      2. Ensure the token is updated before the expiry date to maintain uninterrupted access.
   2. **Steps to Retrieve a New Authentication Token**
      1. Access the Portal:
      2. Open your browser and navigate to: <https://icarus.lums.edu.pk/>.
      3. **Log In:**
         1. Username: Muhammad\_waqar@lums.edu.pk
         2. Password: Waqar123
   3. **Navigate to Security Settings:**
      1. Once logged in, click on the Account option in the upper-right corner of the portal.
      2. From the dropdown menu, select Security.
   4. **Copy the JWT Token:**
      1. Locate the section labeled JWT Token.
      2. Copy the token displayed.
   5. **Update the Token:**
      1. Replace the old token in your application or configuration with the newly retrieved token.
      2. Change this token in below file in request header
         1. Core/management/update\_weather\_station.py
         2. Core/models.py
         3. Cornjobs/weather\_station.py
   6. **Reminder**
      1. Set a reminder to update the token again before 2025-03-16 to avoid any disruptions.
      2. Let me know if you need further clarification or assistance!
4. **parse\_kmz.py:** The command processes a .kmz file to extract geospatial data (polygons and points), identifies cameras within a 10km radius of these geospatial features, and updates related events in the database. For polygons: Check for a minimum of 4 coordinates, calculate the centroid, and find cameras within a 10 km radius. For points: Calculate distances between the point and camera locations to find those within 10 km. For cameras identified in proximity to the geospatial features(fire), find Event objects created on the same day and update their nasa\_tag field to True.