**📄 Project Requirements – Daily Weather Report Email Scheduler**

**1. Project Overview**

We are building a Django application that:

* Lets users **subscribe** to daily weather reports by providing their email and selecting a city from a predefined list.
* At **7:00 AM server time daily**, the system will:
  1. Fetch weather data for all subscribed cities from a free weather API (e.g., OpenWeatherMap).
  2. Send each subscriber an email with their city’s weather details.
  3. *(Optional)* Log this weather data in the database for historical viewing.

**Scheduling Options:**

* **Preferred:** Celery + Redis + django-celery-beat for scheduled jobs.
* **Alternative:** Django management command executed via a system cron job.

**2. Database Models**

**2.1 UserSubscription**

Stores user subscription details.

* **email** → Email address of the subscriber (must be valid format).
* **city** → Name of the city (must be from a predefined list).
* **subscribed\_at** → Date & time when the subscription was created.
* **(Optional)** status → Active/Inactive (if you want soft unsubscribe instead of delete).

**2.2 WeatherLog *(Optional – for history)***

Stores daily weather information for each city.

* **city** → Name of the city.
* **temperature** → Temperature in °C.
* **humidity** → Humidity percentage.
* **conditions** → Short weather description (e.g., “clear sky”).
* **date** → Date when the weather data was recorded.

**3. APIs to be Created**

**3.1 Subscribe API *(Required)***

* **Method:** POST /api/subscribe/
* **Purpose:** Allow users to subscribe to weather reports.
* **Request Fields:** email, city.
* **Validations:**
  + Email format check.
  + City must be in allowed list.
  + Prevent duplicate (email + city) subscriptions.

**3.2 Unsubscribe API *(Required)***

* **Method:** POST /api/unsubscribe/
* **Purpose:** Allow users to remove their subscription.
* **Request Fields:** email, city.

**3.3 List Subscriptions API *(Optional)***

* **Method:** GET /api/subscriptions/?email=user@example.com
* **Purpose:** Retrieve a list of all subscriptions for a given email (for UI or admin use).

**3.4 Weather History API *(Optional)***

* **Method:** GET /api/weather-history/?city=London
* **Purpose:** Return historical weather data for a city from the WeatherLog table.

✅ **API Summary:**

* **Required:** 2 (Subscribe, Unsubscribe)
* **Optional:** 2 (List Subscriptions, Weather History)
* **Total Possible:** 4

**4. Core Functional Requirements**

**4.1 Subscription**

* User can subscribe via a form (HTML) or API.
* Must validate inputs before saving.
* No duplicates allowed.
* Users can unsubscribe via a form, API, or unsubscribe link in the email.

**4.2 Weather Data Fetching**

* Use OpenWeatherMap API:

bash

CopyEdit

GET http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API\_KEY}&units=metric

* Extract temperature, humidity, and conditions.
* Handle API failures gracefully (log & skip).

**4.3 Daily Email Sending**

* Email sent **once per subscriber per day**.
* Subject: Your Daily Weather Report for [City].
* Body: Both HTML and plain text formats.
* Retry failed sends if using Celery.

**4.4 Scheduling**

**Option 1 – Celery + Redis:**

* Use celery-beat to schedule at 7:00 AM daily.
* Workflow:
  1. Get unique cities from subscriptions.
  2. Fetch weather for each city once.
  3. Save to WeatherLog (optional).
  4. Send emails to all subscribers for that city.

**Option 2 – Cron + Management Command:**

* Create weather\_task command.
* Cron triggers:

pgsql

CopyEdit

0 7 \* \* \* /path/to/python manage.py weather\_task

**4.5 Optional – Weather History**

* Store each city’s daily weather in WeatherLog.
* Allow querying via API for historical weather.

**5. Non-Functional Requirements**

* **Scalability:** Fetch weather per city once.
* **Reliability:** Log all failures (API, email).
* **Performance:** Use bulk queries.
* **Security:** Validate and sanitize inputs.
* **Configuration:** API key, SMTP, schedule time via environment variables.

**6. Execution Flow**

**Step 1 – User Subscription**

1. User visits form/API.
2. Enters email & city.
3. Backend validates and saves.
4. User gets success message.

**Step 2 – Daily Scheduled Task**

1. Scheduler triggers at 7:00 AM.
2. Fetch weather for each city.
3. Send personalized emails to subscribers.
4. Log results.

**Step 3 – Optional History Query**

1. User/API requests history for a city.
2. Return records from WeatherLog.

**7. Developer Deliverables**

1. Django project with models, views, APIs.
2. Weather fetch service.
3. Email service (HTML + plain text).
4. Celery or cron setup.
5. Error logging & retries.
6. README with setup & usage instructions.

**🧰 Git & Version Control Requirements:**

1. Initialize a Git repository at the very beginning of the project.
2. Create a .gitignore file to exclude unnecessary files (e.g., .pyc, \_\_pycache\_\_, venv, db.sqlite3, etc.).
3. Host the code in a remote GitHub repository. Name it clearly
4. Push the project to GitHub immediately after project setup (before writing full code).
5. Make regular commits with meaningful messages.
6. Do not wait until the end to push code. Push updates frequently (e.g., after completing a model, view, or template section).
7. Organize work using separate branches if possible (optional but encouraged).
8. Include a README.md file with setup instructions and project details.

**📦 Environment Setup (requirements.txt):**

* Create a requirements.txt file that includes all required packages to run the project.
* It must include at least: