## **Ссылка на проект:** [**https://github.com/NikkZav/open-meteo\_API**](https://github.com/NikkZav/open-meteo_API)

## **Описание методов API**

### **1. GET /weather**

**Описание:**Метод принимает координаты (широту и долготу) и возвращает данные о температуре, скорости ветра и атмосферном давлении на текущий момент.  
**Принцип работы:**

* Координаты передаются через query‑параметры (схема Coordinates).
* Сначала сервис WeatherService пытается найти погодные наблюдения в базе данных на основе координат (сделал для оптимизации, возможно это лишнее). И только если записи отсутствуют, то тогда происходит непосредственное обращение к внешнему API Open‑Meteo для получения актуального прогноза.
* Ответ формируется согласно параметрам, заданным через схему WeatherQueryParams, и возвращается в формате WeatherResponse.  
  (в ТЗ для этого метода были конкретные требования к возвращаемым значениям “данные о температуре, скорости ветра и атмосферном давлении”, но т.к. потом же упоминается, что “должна быть возможность выбирать какие параметры погоды получаем в ответе” решил добавить эту возможность везде)

### **2. POST /add\_city**

**Описание:**Метод принимает название города и его координаты (через схему CityParams) и добавляет город в базу данных для отслеживания прогноза погоды.  
**Принцип работы:**

* Сначала сервис CityService проверяет уникальность города по имени и координатам. Если город с такими данными уже существует, генерируется исключение, которое возвращает HTTP‑409.
* При успешном прохождении проверок, сервис запрашивает начальные погодные данные (через WeatherRepository) и сохраняет новый город с соответствующими записями погоды в базе.
* После сохранения нового города запускается фоновая задача (через create\_periodic\_weather\_update\_task), которая периодически обновляет погодные данные для этого города. Обновление реализовано именно таким образом (НЕ единовременное для всех городов, хотя так было бы оптимальнее с точки зрения кол-ва запросов в БД), т.к. всё запускается одной точкой входа (script.py) и не предполагает использования внешних фоновых механизмов, таких как Celery, cron или специализированные планировщики задач ( + по почте ответили, что такой механизм предпочтительнее, как я понял).
* В ответ будет сообщение об успешном добавлении, id и название города.

### **3. GET /cities**

**Описание:**Метод возвращает список городов, для которых доступен прогноз погоды. Опционально можно вернуть города с вложенными погодными данными.  
**Принцип работы:**

* Сервис CityService использует репозиторий CityRepository для получения списка городов. Если в запросе установлен флаг include\_weather, то для каждого города дополнительно предзагружаются связанные записи погоды.  
  (Этого не было в ТЗ, но решил опционально реализовать такой функционал, надеюсь инициатива не наказуема, в данном случае. О проблеме перегруженного ответа огромным количеством данных знаю, но в связи с отсутствием понимания как именно и где эта API может применяться, решил оставить, по крайней мере теперь есть возможность удобно просматривать данные из БД, что надеюсь поможет при проверке).
* Результат преобразуется в формат, соответствующий схеме CityResponse, и возвращается клиенту. (тут конечно из-за вышеописанной функциональности не очень красиво работает возвращаемые формат и в реальном проекте я бы не стал так делать, но всё-таки решил оставить такую функциональность).

### **4. GET /weather/{city\_name}**

**Описание:**Метод принимает название города и время и возвращает для этого города прогноз погоды на текущий день, ближайший к указанному времени.  
**Принцип работы:**

* Название города передается как часть URL, время — через query‑параметр.
* Сначала проверяется, что указанное время соответствует сегодняшнему дню. Если нет, генерируется ошибка (TimeRangeError) и возвращается HTTP‑400.
* Сервис WeatherService ищет город по имени, затем пытается найти в базе данных погодные записи для этого города. Если записи есть, выбирается запись, время которой максимально близко к запрошенному.
* Если погодные данные отсутствуют в БД, происходит обращение к внешнему API Open‑Meteo.
* Ответ формируется с учётом параметров запроса (через WeatherQueryParams) и возвращается в формате WeatherResponse.

К сожалению “дополнительные задания” я попросту не успел реализовать, в связи с некоторой загруженностью именно в момент выполнения данного тестового задания, но я постарался сделать упор именно на архитектуру и общую “чистоту” кода, хотя конечно тут много чего стоит переработать и дописать (как минимум док-стринги везде и унифицировать всё под английский язык, но решил оставить русский для удобства проверяющих). Надеюсь это скромная, но честная работа, продемонстрирует необходимые навыки для стажёра, который максимально заряжен на работу и учёбу!