Design specification

Interview exercise

Revision history

Date	Updated by	Notes
18-SEP-2025	Gastón Ortiz	Initial version

Basic information

The project contains a single endpoint that retrieves weather data for a city. There are some restrictions applied to it:

1. The city must be within a fixed set of cities handled by the application. As a potential extension to the application, dynamically defining this list is recommended.
2. The city must have a IATA code.
3. Weather data retrieved is limited to temperature, humidity and broad description.

Weather retrieval endpoint data

<table>
<tr><td><b>Relative path</b></td><td colspan="2">/weather/{city_code}</td></tr>
<tr><td><b>Parameter</b></td><td><b>Type</b></td><td><b>Description</b></td></tr>
<tr><td>city_code</td><td>Path parameter (String)</td><td>The IATA city code for the city whose weather we are looking for.
The `city_codes.csv` file contains the accepted city codes and its matching name.
Default accepted values are: NYC, LON, PAR, BUE, RIO, TYO, CHI, MOW, ROM, MIL, BER, SAO, IST, DEL, BOM, YTO, YMQ, SFO, LAX, WAS, BJS, SHA, SEL, OSA, UKY</td></tr>
</table>

Weather retrieval return types

The endpoint returns an instance of the Weather object converted to JSon.

Object name	Weather	
Attribute	**Type**	**Description**
cityCode	CityCode	The object matching the City Code of the weather sought.
timestamp	LocalDateTime	The timestamp of the operation.
report	Report	The weather report containing the information retrieved from Weather API.

Object name	CityCode	
Attribute	**Type**	**Description**
code	String	The IATA code for the city.
name	String	The name of the city.

Object name	Report	
Attribute	**Type**	**Description**
temp	double	The latest temperature in ºC registered in the city.
description	String	The brief description of the weather (i.e. Sunny)
humidity	int	The latest percentage of humidity registered in the city.

Weather retrieval error codes

When an exception occurs, the endpoint will return the appropriate status code containing as the response body a JSon with a representation of the following object:

Object name	ApplicationError	
Attribute	**Type**	**Description**
code	ErrorCode (Enum)	The internal error code identifier.

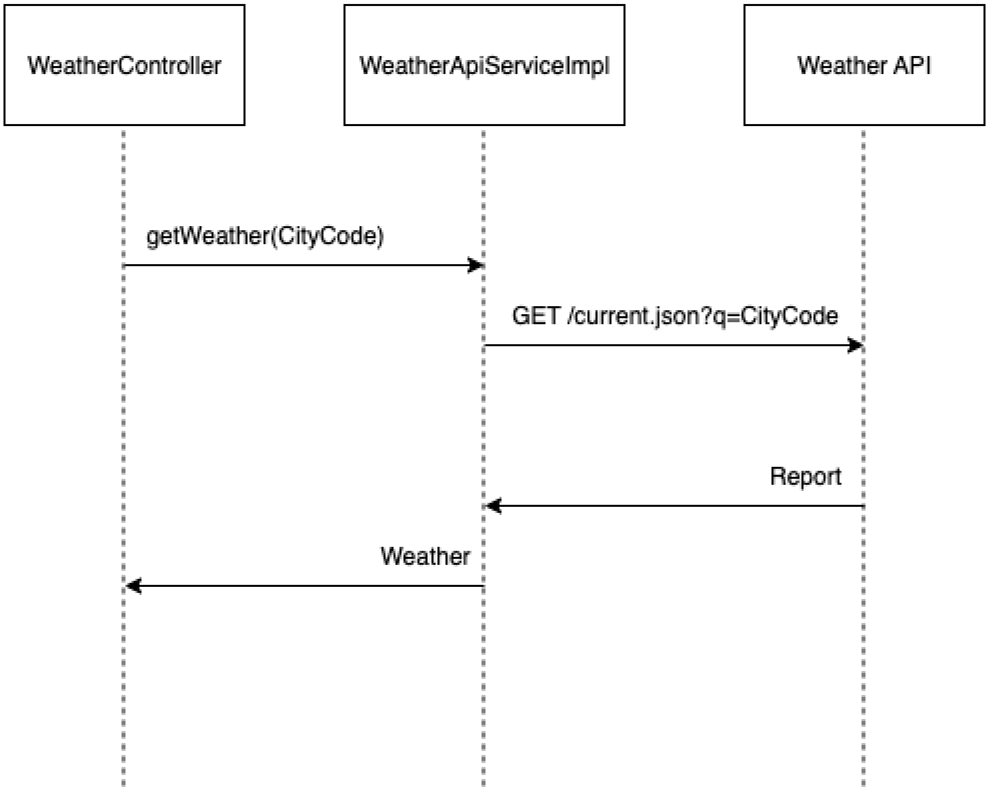
message	String	The message of the error displayed to the user.

Error code list

The following is a list of error codes and their description:

Code	**Associated status code**	**Description**
INVALID_CITY_CODE	400 (Bad request)	The passed city code is not within the city codes list.
CITY_NAME_MISCONFIGURATION	500 (Internal server error)	The city_codes.csv file containing the name configurations is matching an invalid city name to a IATA code.
API_NOT_FOUND	500 (Internal server error)	The url set is not a valid Weather API endpoint.
MISSING_API_KEY	500 (Internal server error)	The API key configuration variable hasn’t been set.
INVALID_API_KEY	500 (Internal server error)	The API key configured is invalid.
EXCEEDED_API_CALLS_LIMIT	403 (Forbidden)	The amount of Weather API invocation has exceeded the daily limit.
DISABLED_API_KEY	403 (Forbidden)	The API key configured is disabled.
UNKNOWN_ERROR	500 (Internal server error)	An unexpected error has occurred.

Process flow



Processing details

The flow is quite simple: The controller received a String containing a IATA code for a city, which it validates and converts to a CityCode object that then passes to the WeatherService implementation to retrieve the weather from. In turn, the service gets the name from the CityCode and uses it as a parameter to do a request to the WeatherAPI API and gets a Report instance from it, parses it into a Weather by adding the location and time context and then returns it into the controller, who then converts it into JSon and returns it to the user.

Validation & conversion

Before reaching the controller, the input String is converted to a CityCode object by using the CityCodeConverter. It loads a Map<String, CityCode> at startup from city_codes.csv (classpath) and converts input code (case-insensitive) to a CityCode using it. Any missing or unknown codes will return an exception to the user.

Internal methods

Component	Signature / Type	Responsibility
`WeatherController`	`Weather getWeatherByCity(@PathVariable("code") CityCode cityCode)`	HTTP entrypoint; logs and delegates to service.
`WeatherService`	`Weather getWeather(CityCode cityCode)`	Interface for weather retrieval.
`WeatherApiServiceImpl`	`Weather getWeather(CityCode cityCode)`	Calls WeatherAPI via `HttpClientService`, maps `Report` → `Weather`, wraps API errors to `CodedException`.
`HttpClientService`	`<R,E> R get(String path, Map<String,String> q, Class<R> ok, Class<E> err)`	Builds URL with default + request params, executes OkHttp, parses success or error payloads.
`CityCodeConverter`	`CityCode convert(String code)`	Validates/normalizes IATA city code using startup cache from CSV.
`ReportDeserializer`	`JsonDeserializer<Report>`	Extracts `temp_c`, `condition.text`, `humidity` from WeatherAPI JSON `current` node.
`ErrorDeserializer`	`JsonDeserializer<ApplicationError>`	Maps WeatherAPI error codes to internal `ErrorCode` enum and message.
`GlobalExceptionHandler`	`@ExceptionHandler(...)`	Maps domain and framework exceptions to HTTP responses with `ApplicationError`.

`ApplicationError`	`record ApplicationError(ErrorCode code, String message)`	Uniform error body; `@JsonDeserialize(using = ErrorDeserializer.class)`.
`CityCode`	`record CityCode(String code, String name)`; `CityCode(Map<String,String> row)`	Domain for city codes; map ctor supports CSV rows.
`Weather`	`record/class Weather(CityCode cityCode, LocalDateTime timestamp, Report report)`	API response model (temp/description/humidity derived from `Report`).
`Report`	`record/class Report(double temp, String description, int humidity)`	Internal parsed payload from WeatherAPI’s `current` section.