# Weather Data Caching with Redis (Cache-Aside and Write-Through Mechanisms)

## Project Goal

Develop a caching layer for weather data using Redis to boost API performance. We apply cache-aside and write-through mechanisms to handle data retrieval and updates, improving response times and reducing primary data source load.

## Theory

### 1. Caching Overview

Caching helps store frequently accessed data temporarily in a high-speed storage layer to reduce latency and improve system performance. In this project, we use Redis as a cache store for weather data.

### 2. Cache Mechanisms Used

- Cache-Aside (Lazy Loading): Checks Redis for requested data before querying the primary data source. If data is absent, it fetches data from the primary source, then stores it in the cache.

- Write-Through: Any updates to data (e.g., weather data) are immediately reflected in the cache to keep data consistent.

## Project Structure and Components

1. Main Class (WeatherApplication.java): Uses @EnableCaching to activate caching, allowing Spring to process annotations like @Cacheable and @CachePut.  
2. Controller (WeatherController.java): Defines API endpoints for retrieving and updating weather data.  
3. Service (WeatherService.java): Implements business logic to manage caching.

## Caching Annotations in Use

- @EnableCaching: Enables caching in Spring Boot to allow the caching mechanisms to work.  
- @Cacheable: Applied to the data-fetching method. Spring first checks the Redis cache; if no data is found, it retrieves the data from the primary source and caches it.  
- @CachePut: Used to update the cache when data is modified, ensuring that the cache remains consistent with the source data.

## Cache Flow and Mechanisms

1. Data Retrieval Flow (Cache-Aside):  
- When the GET /api/weather/{city} endpoint is called:  
 \* Spring checks Redis for cached data (cache-aside).  
 \* If data exists in Redis, it’s returned instantly.  
 \* If data isn’t cached, the service fetches it from the primary source (e.g., static values) and caches it in Redis.

2. Data Update Flow (Write-Through):  
- When the POST /api/weather/{city}?weather=<newWeather> endpoint is called:  
 \* The @CachePut annotation refreshes the cache with new data, ensuring consistency between Redis and the source.  
 \* This write-through approach keeps Redis up-to-date with the latest values.

## Project Execution and Output

* Example Requests and Outputs:

1. 1. Fetching Weather Data (Cache-Aside):

curl http://localhost:7001/api/weather/NewYork  
 Output:  
 Sunny

1. 2. Updating Weather Data (Write-Through):

curl -X POST "http://localhost:7001/api/weather/NewYork?weather=Cloudy"  
 Output:  
 Cloudy

* Verifying Cache in Redis:

Use the redis-cli to check for cached entries.  
redis-cli  
127.0.0.1:6379> keys \*  
1) "weatherCache::NewYork"

## Redis CLI Sample Output

root@Adira:~# redis-cli  
127.0.0.1:6379> ping  
PONG  
127.0.0.1:6379> keys \*  
(empty array)  
127.0.0.1:6379> keys \*  
1) "weatherCache::NewYork"  
127.0.0.1:6379> keys \*  
1) "weatherCache::NewYork"