# Interactive Project Documentation: Advanced Caching in Spring Boot with Redis

\*\*Goal\*\*: Implement an advanced caching mechanism using Spring Boot with Redis, demonstrating both Cache-Aside and Write-Through strategies.  
This project helps reduce load on the primary data source, improves response time, and illustrates when to use caching vs. accessing the main data source directly.

## Theory and Concepts

1. \*\*Caching in Distributed Systems\*\*: Caching is essential for high-performance applications. It reduces load on primary data sources by storing frequently accessed data in a fast storage layer.  
  
2. \*\*Cache-Aside Pattern\*\*: This pattern loads data into the cache only upon a cache miss. The application retrieves data from the primary source and stores it in the cache for future use.  
  
3. \*\*Write-Through Pattern\*\*: Updates are written to both the cache and the primary data source at the same time. This pattern ensures data consistency between the cache and the database.  
  
4. \*\*TTL (Time-to-Live)\*\*: Setting TTL ensures cached data expires after a specific time, allowing it to stay fresh.

## Project Structure and Implementation Flow

1. \*\*WeatherService Class\*\*:  
 - Provides a default weather output as a fallback in case data is not found in the cache.  
 - Simulates primary data source retrieval.  
  
2. \*\*CacheService Class\*\*:  
 - `getWeather`: Uses @Cacheable to implement Cache-Aside caching. It retrieves data from the cache if present, or from the primary source if missing.  
 - `updateWeather`: Uses @CachePut for Write-Through caching, ensuring the cache and primary data source are both updated.  
  
3. \*\*Controller\*\*:  
 - Manages API requests for getting and updating weather data.  
 - Routes calls to the respective service methods based on cache status.

## Code Flow

1. \*\*Cache Miss\*\*: If Redis cache is empty, the application checks the primary source (WeatherService) for data.  
2. \*\*Cache Hit\*\*: If data is present in Redis, it is returned directly.  
3. \*\*Updating Data\*\*: Updates apply simultaneously to both cache and primary source (Write-Through), ensuring data consistency.

## Example Outputs

- \*\*Postman Request\*\*:  
 - GET `http://localhost:7002/api/cache/weather/NewYork` -> "Weather in NewYork: Sunny"  
 - POST `http://localhost:7002/api/cache/weather/NewYork?weather=Cloudy` -> "Weather updated to Cloudy"  
 - GET after update -> "Weather in NewYork: Cloudy"  
  
- \*\*Redis CLI\*\*:  
 - `redis-cli keys \*` -> Shows cache key e.g., "weatherCache::NewYork"