**Cache-service-redis**

* A redis plugin for cache-service
* AND a standalone redis microservice

# What does cache-service do?

cache-service allows you to create redundant, cache-agnostic caching configurations. I've supplied a redis dB controller and redis-cache controller separately(just to test caching), but there can be any datascource you want as long as you follow the same interface.

# Install

* Download [Redis-x64-2.8.2104.zip](https://github.com/MSOpenTech/redis/releases/tag/win-2.8.2104)
* Extract the zip to prepared directory
* run redis-server.exe
* then run redis-cli.exe
* You can start using Redis now please refer for [commands](http://redis.io/commands)

# The Redis Configuration

|  |
| --- |
| To define the connection settings between the application client and the Redis server instance, we need to use Redis client.  There is a number of Redis client implementations available for Java. In this implementation Jedis- a simple and a powerful Redis client implementation is used.  There is good support for both XML and Java configuration in the framework, in this implementation Java-based configuration is done.  @Bean  JedisConnectionFactory jedisConnectionFactory() {      return new JedisConnectionFactory();  }    @Bean  public RedisTemplate<String, Object> redisTemplate() {      RedisTemplate<String, Object> template = new RedisTemplate<>();      template.setConnectionFactory(jedisConnectionFactory());      return template;  }  The configuration is quite simple. First, using the Jedis client, we’re defining aconnectionFactory.  Then, we defined a RedisTemplate using the jedisConnectionFactory. This can be used for querying data with a custom repository. |
|  |

The Redis configuration is used here as there’s a Redis Datascore in this springboot application just to test the redis caching service (this caching service can be integrated with any datasoucre).

# Configuring Redis Cache

With Spring Boot and the required dependency already in work with Maven, we can configure local Redis instance with only three lines in our application. Properties file as:

#Redis config

spring.cache.type=redis

spring.redis.host=localhost

spring.redis.port=6379

Also, @EnableCaching annotation has to be used Spring Boot main class

# Example

}

// Controller Code under test

@Cacheable(value = "examples", key = "#id")

@GetMapping("/getExampleById/{id}")

**public** Example getExample (@PathVariable String id) **throws** CachingException {

logger.info("Getting example with ID {}.", id);

**Return** exampleCachingService.getExample(id);

}

// Test

@Test

**public** **void** testGetExample () **throws** CachingException {

Example example = **new** Example();

*when*(exampleServiceImpl.getExample("1")).thenReturn(example);

*assertEquals*(example, exampleCachingController.getExample("1"));

}

// Service Code under test

@Override

**public** Example getExample (String id) **throws** CachingException {

logger.info("Getting example with ID {}.", id);

Optional<Example> customer = ExampleDAO.findById(id);

**if** (example.isPresent()) {

logger.info("from database>>>>>>>>>>>>>>>>>>>>>>>>>>>>");

**return** example.get();

} **else** {

**throw** **new** CachingException(environment.getProperty("200"));

}

}

// Test

@Test

**public** **void** testGetExample() **throws** CachingException {

Customer example = **new** Example();

example.setId("1");

example.setName("Redis-Cache Example");

Optional<Customer> exampleOpt = Optional.*of*(example);

*when*(exampleDAO.findById("1")).thenReturn(exampleOpt);

*assertEquals*(example, exampleCachingServiceImpl.getExample("1"));