

Redis, geliştiriciler tarafından en çok kullanılan ve bilinen NoSQL veritabanlarından birisidir. Redis, açık kaynaktır ve kaynak kodlarına GitHub üzerinden erişilebilmektedir.

**Redis** – Remote Dictionary Server (Uzak Sözlük Sunucusu); ilişkisel olmayan anahtar/değer veri tabanlarını ve önbellekleri uygulamak için yaygın olarak kullanılan açık kaynaklı bir bellek içi veri deposudur.

C dili ile yazıldığı için yüksek performanslı sonuçlar vermektedir.

Linux ve türevi işletim sistemleri tarafından desteklenmekte fakat Windows tarafı için resmi bir destek olmasa da community tarafından desteklenmektedir.

Redis günümüz sistemlerinde en çok kullanılan key-value veritabanıdır ve genellikle ***caching, session yönetimi, pub/sub, message broker*** amacıyla kullanılmaktadır.



## Redis'in Avantajları

### Yüksek Performans

Redis, verileri disklerde (HDD veya SSD) tutan veri tabanlarının aksine bellek (RAM) üzerinde tutar bu sayede disklere erişim ihtiyacını ortadan kaldırarak gecikmeleri, I/O bağlantılarını önler ve daha az CPU kullanan basit algoritmalar ile verilere erişir.

### In-Memory Veri Yapıları

Redis verileri bellek üzerinde <key,value> çifti olarak tutmaktadır, burada herbir anahtara denk gelen değerler farklı veri yapılarında tutulabilmektedir. Bu veri yapıları; **String, List, Hash, Set, Sorted Set, Bitmaps, HyperLogLogs, Geospatial Indexes**

Redis kullanılarak neredeyse her türlü veri bellekte saklanabilir.

### Replication

Redis, master-slave mimarisini kullanır, master genel olarak yazma işlemlerini yapar ve slave dediğimiz yapılar da master'in birer kopyasıdır, master güncellendikçe ona bağlı bütün slave'ler de güncellenir. Burada master'da oluşacak herhangi bir çökmede, hatada direkt bir slave master olarak seçilir ve sistem çalışmaya devam eder.

### Persistence (Veri Kalıcılığı)

Redis'te verilerin RAM üzerinde saklandığından bahsettik, olası bir elektrik kesintisi, sunucu kapanması gibi durumlarda veriler silinecektir. Redis bize iki yöntem sunmaktadır verinin kalıcılığını sağlamak için. Bunlar; **point-in-time Snapshots ve Append Only File (AOF)**.

**Snapshots** yönteminde belirli zaman aralıkları ile RAM üzerindeki verinin kaydı, kopyası diske kayıt edilir bu sayede olası bir elektrik kesintisi gibi durumlarda disk üzerinden verilere tekrar geri dönülebilir.

**Append Only File** yönteminde ise her değişikliği dosyanın sonuna yazarak oluşan veri değişikliklerinin kaydını tutar.

### Çoklu Dil Desteği

Redis birçok dil tarafından desteklenmektedir, bunlar; Java, Python, PHP, C, C ++, C #, JavaScript, Node.js, Ruby, R, Go gibi dillerdir ve bunların yanı sıra daha fazla da dil bulunmaktadır.

## Redis’in Bazı Kullanım Senaryoları

### Caching (Önbellek) Mekanizması

Sıkça kullanılan verilerimizi sürekli veritabanına ya da diğer kaynaklara gidip çağırmak yerine ön belleğe almak performans açısından olumlu bir katkı sağlayacaktır. Dağıtık mimaride çalışabilen Redis, dağıtık cache(distributed caching) yönetimi için birçok uygulamada kullanılmaktadır.

### Session Yönetimi

Uygulamalarımızı kullanırken kullanıcılara ya da diğer yapılara ait verilerimizi sayfalar arasında taşımak için session’dan sıkça yararlanmaktayız fakat uygulamamız büyüdükçe bu verilerin tutulması artan bellek alanına neden olmaktadır. Redis; sosyal medya, e-ticaret uygulamaları, oyun gibi alanlarda session bilgilerinin tutulmasında da rol almaktadır.

### Pub/Sub

Redis, pub/sub işlevini destekleyen komutlara da sahiptir ve Redis’in broadcast yayını yapmasına olanak sağlar. Bu, mesajı tek bir istemcinin bir kanala bağlı diğer birçok istemciye yayınlamasına olanak tanır.

### Queues (Kuyruklar)

Redis, gerçekleşmesi zaman alacak işleri bir kuyruk yapısına alınmasını ve daha sonradan işlenmesini destekler.

## Birçok Veri Türü

Redis, birçok farklı veri türünü sunabilen bir NoSQL veritabanı çözümüdür ayrıca bu veri türlerinden en iyi şekilde yararlanmak için ihtiyacınız olan komutları da sağlar. Aşağıda Redis’e ait bazı veri türlerini ve bu türlere ait temel terminal komutlarını görebilirsiniz.

### Strings

Verilerin string formatı olarak tutulduğu veri yapısıdır. <key,value> çiftinde değer kısmı maksimum 512 MB yer tutar.

String veri ekleme komutu:

```
SET <key> <value>
```

```
127.0.0.1:6379> SET firstName Yusuf
OK
```

Anahtar değeri ile string veriyi alma komutu:

```
GET <key>
```

```
127.0.0.1:6379> GET firstName
"Yusuf"
```

Değerin sonuna string ekleme komutumuz:

```
APPEND <key> <text>
```

```
127.0.0.1:6379> APPEND firstName " Yilmaz"
(integer) 12
127.0.0.1:6379> GET firstName
"Yusuf Yilmaz"
```

Bütün anahtar değerlerini çağırma komutu:

**KEYS \***

```
127.0.0.1:6379> KEYS *
1) "firstName"
```

Anahtar değeri ile veri silme komutu:

**DEL <key>**

```
127.0.0.1:6379> DEL firstName
(integer) 1
```

Bütün verileri silme komutu:

**FLUSHALL**

```
127.0.0.1:6379> FLUSHALL
OK
```

Yukarıdaki komutlarda her ne kadar string işlemleri yapsak da sayılsa bir değeri string gibi tutup üzerinde matematiksel işlemler de gerçekleştirebiliriz.

Sayısal değeri bir artırma komutu:

**INCR <key>**

```
127.0.0.1:6379> SET age 22
OK
127.0.0.1:6379> INCR age
(integer) 23
```

Yukarıdaki INCR komutu ile değer bir artırıldı. Bunun yanı sıra **DECR** , **DECRBY** , **INCRBY** gibi komutlar ile de artırım ya da azaltım işlemleri yapılabilmektedir.

Oluşturulacak değere yaşam süresi ekleme komutu:

**SETEX <key> <expire\_time> <value>**

```
127.0.0.1:6379> SETEX lastName 20 Yilmaz
OK
127.0.0.1:6379> TTL lastName
(integer) 19
```

**TTL** komutu verilen anahtar değerin süresinin bitmesine kaç saniye kaldığını söyler, verilen **expire\_time** değeri saniye cinsindendir.

## Lists

Burada veriler bağlı liste biçiminde tutulurlar.

Bu listede, liste başını ifade etmek için L, liste sonunu ifade etmek için de R kullanılır.

Listenin başına eleman ekleme komutu:

```
LPUSH <key> <value>
```

```
127.0.0.1:6379> LPUSH postList "Dependency Injection ve Ninject"  
(integer) 1
```

Listenin sonuna eleman ekleme komutu:

```
RPUSH <key> <value>
```

```
127.0.0.1:6379> RPUSH postList "Aspect Oriented Programming"  
(integer) 2
```

Listenin eleman sayısını öğrenme komutu:

```
LLEN <key>
```

```
127.0.0.1:6379> LLEN postList  
(integer) 2
```

Listenin başından eleman silme komutu:

```
LPOP <key>
```

```
127.0.0.1:6379> LPOP postList  
"Dependency Injection ve Ninject"
```

Listenin sonundan eleman silme komutu:

```
RPOP <key>
```

```
127.0.0.1:6379> RPOP postList  
"Aspect Oriented Programming"
```

Listede belirli aralıktaki verileri alma komutu:

```
LRANGE <key> <start_index> <stop_index>
```

```
127.0.0.1:6379> LRANGE postList 0 2  
1) "Fluent Validation"  
2) "Dependency Injection ve Ninject"  
3) "Aspect Oriented Programming"
```

Liste üzerindeki eleman sayısını bilmediğimiz durumlarda `start_index` değerine 0 `stop_index` değerine -1 verirse**k** bütün verileri elde edebiliriz.

## Sets

Set veri tipi, verileri sırasız (rastgele sırada eklenilen) ve unique (benzersiz) olarak tutan veri tipidir. Aynı veriden birden fazla bulunmamaktadır.

Set'e eleman ekleme komutu:

```
SADD <key> <value>
```

```
127.0.0.1:6379> SADD friends Halit
(integer) 1
```

Set içerisinden eleman silme komutu:

```
SREM <key> <value>
```

```
127.0.0.1:6379> SREM friends Halit
(integer) 1
```

Set içerisindeki bütün elemanları getirme komutu:

```
SMEMBERS <key>
```

```
127.0.0.1:6379> SMEMBERS friends
1) "Arafat"
2) "Halit"
```

## Sorted Sets

Sorted Set veri yapısı, Set veri yapısının benzeridir. Verileri unique (benzersiz) olarak tutmakla beraber **score** dediğimiz değere göre sıralama işlemi yapmaktadır.

Sorted Set'e eleman ekleme komutu:

```
ZADD <key> <score> <value>
```

```
127.0.0.1:6379> ZADD teams 100 "Besiktas"
(integer) 1
```

Sorted Set içerisinden eleman silme komutu:

```
ZREM <key> <value>
```

```
127.0.0.1:6379> ZREM teams Besiktas
(integer) 1
```

Sorted Set içerisindeki belirli aralıktaki verileri alma komutu:

```
ZRANGE <key> <start_index> <stop_index>
```

```
127.0.0.1:6379> ZRANGE teams 0 1
1) "Fenerbahce"
2) "Galatasaray"
```

Sorted Set üzerindeki eleman sayısını bilmediğimiz durumlarda **start\_index** değerine 0 **stop\_index** değerine -1 verirsek bütün verileri elde edebiliriz. Komutun sonuna **WITHSCORES** yazılarak score değerleri ile birlikte veriler elde edilmiş olur.

## Hashes

Hash veri tipi, bir key'e karşılık birden fazla field (alan tutmaya) yarayan veri tipidir.

Hash'e eleman ekleme komutu:

```
HSET <key> <field> <value>
```

```
127.0.0.1:6379> HSET person firstName Yusuf  
(integer) 1
```

Hash içerisinde belirli alanı getirme komutu:

```
HGET <key> <field>
```

```
127.0.0.1:6379> HGET person firstName  
"Yusuf"
```

Hash içerisinde belirli alanı silme komutu:

```
HDEL <key> <field>
```

```
127.0.0.1:6379> HDEL person firstName  
(integer) 1
```

Hash içerisindeki bütün alanı getirme komutu:

```
HGETALL <key>
```

```
127.0.0.1:6379> HGETALL person  
1) "firstName"  
2) "Yusuf"  
3) "lastName"  
4) "Yilmaz"  
5) "age"  
6) "22"
```

Docker Hub'da redis'i bulalım.

[https://hub.docker.com/\\_/redis](https://hub.docker.com/_/redis)

```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~  
$ docker-machine ls  
NAME      ACTIVE  DRIVER      STATE     URL         SWARM      DOCKER      ERRORS  
default   -       virtualbox   Stopped  
  
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~  
$ docker-machine start default  
Starting "default"...  
(default) Check network to re-create if needed...  
(default) Windows might ask for the permission to configure a dhcp server. Sometimes, such confi  
(default) Waiting for an IP...  
Machine "default" was started.  
Waiting for SSH to be available...  
Detecting the provisioner...  
Started machines may have new IP addresses. You may need to re-run the `docker-machine env` com  
  
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~  
$ docker-machine env default  
export DOCKER_TLS_VERIFY="1"  
export DOCKER_HOST="tcp://192.168.99.102:2376"  
export DOCKER_CERT_PATH="C:\Users\EmreGltkn\.docker\machine\machines\default"  
export DOCKER_MACHINE_NAME="default"  
export COMPOSE_CONVERT_WINDOWS_PATHS="true"  
# Run this command to configure your shell:  
# eval $(C:\Users\EmreGltkn\bin\docker-machine.exe env default)  
  
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~  
$ eval "$(docker-machine env default)"
```

```
docker run -p 6379:6379 --name redisserver -d redis
```

```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker run -p 6379:6379 --name redisserver -d redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
b85a868b505f: Pulling fs layer
b09642bd3b88: Pulling fs layer
e0678a951c8d: Pulling fs layer
d5d7c0a1681b: Pulling fs layer
```

```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
7f8cab4b2d81   redis    "docker-entrypoint.s..." About a minute Up About a minute 0.0.0.0:6379->6379/tcp             redisserver
b8b3b9a2c43a   postgres "docker-entrypoint.s..." 7 days ago    Up 8 minutes  0.0.0.0:5432->5432/tcp             resources_db_1
6d4ecef30df5   adminer  "entrypoint.sh docke..." 7 days ago    Up 8 minutes  0.0.0.0:8080->8080/tcp             resources_adminer_1
```

```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker-machine ip
192.168.99.102
```

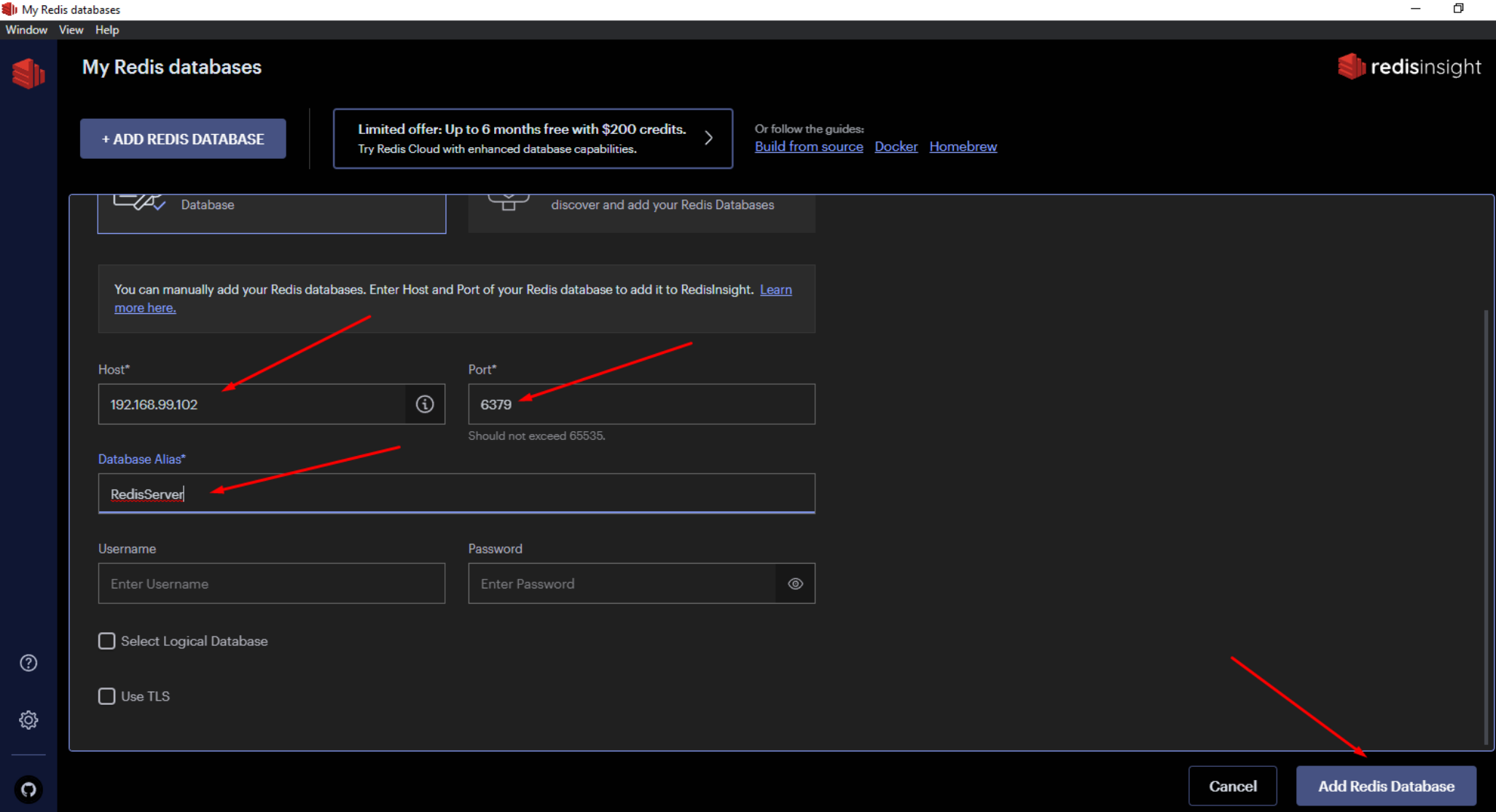
```
winpty docker exec -it redisserver redis-cli
```

```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ winpty docker exec -it redisserver redis-cli
127.0.0.1:6379>
127.0.0.1:6379> |
```

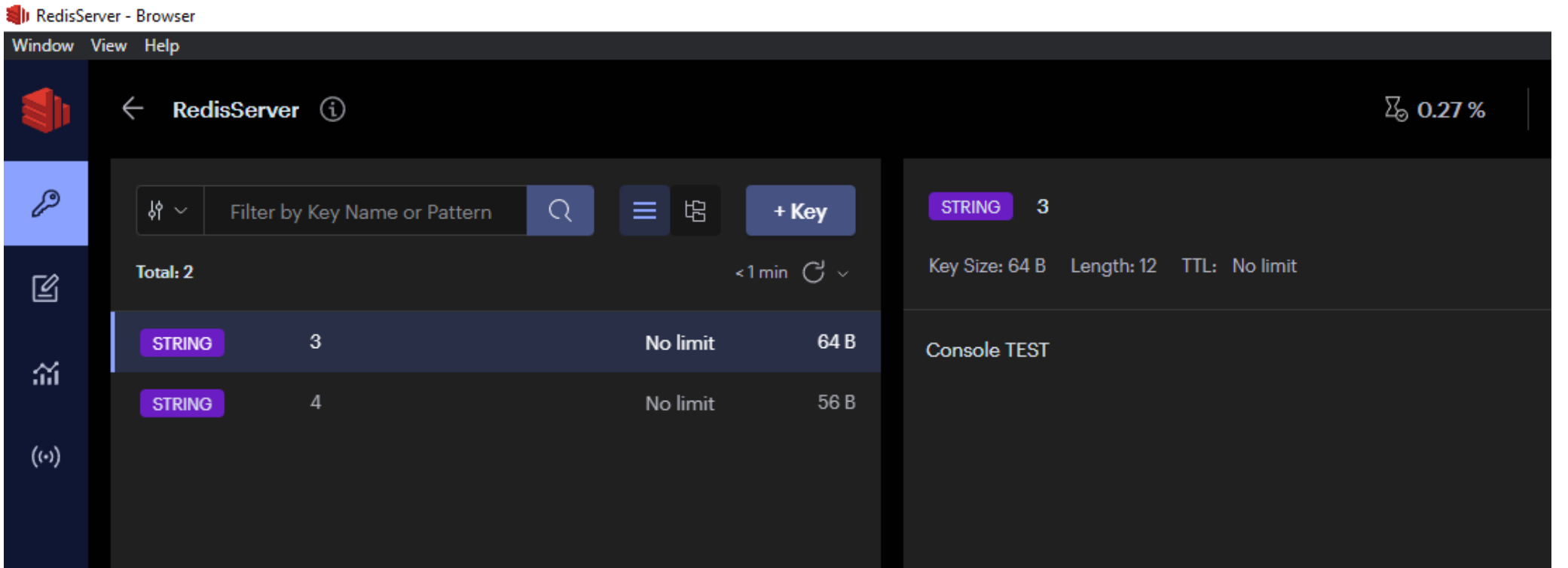
```
127.0.0.1:6379>
127.0.0.1:6379> set 3 "Console TEST"
OK
127.0.0.1:6379>
127.0.0.1:6379> get 3
"Console TEST"
127.0.0.1:6379> |
```

Redis GUI:

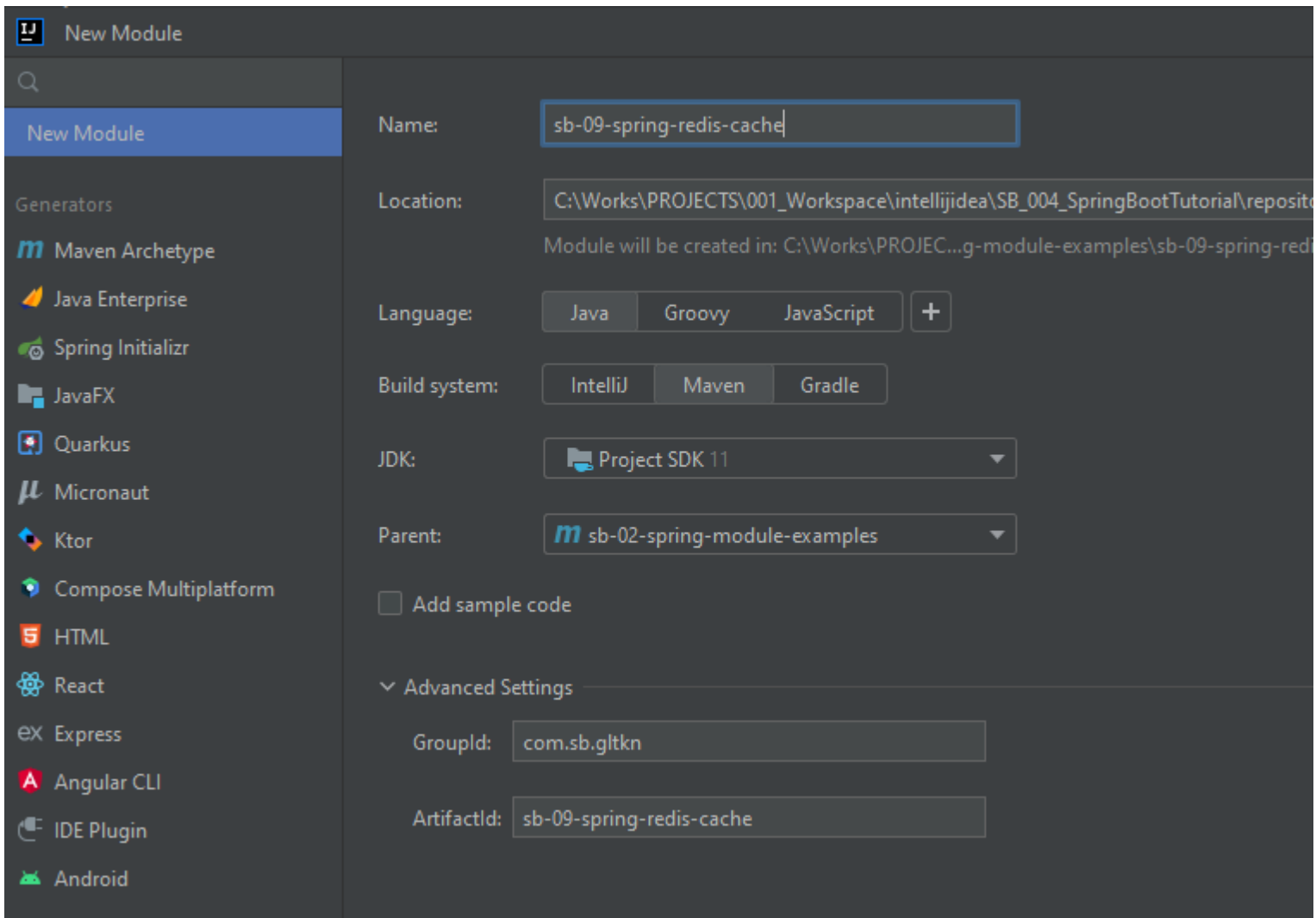
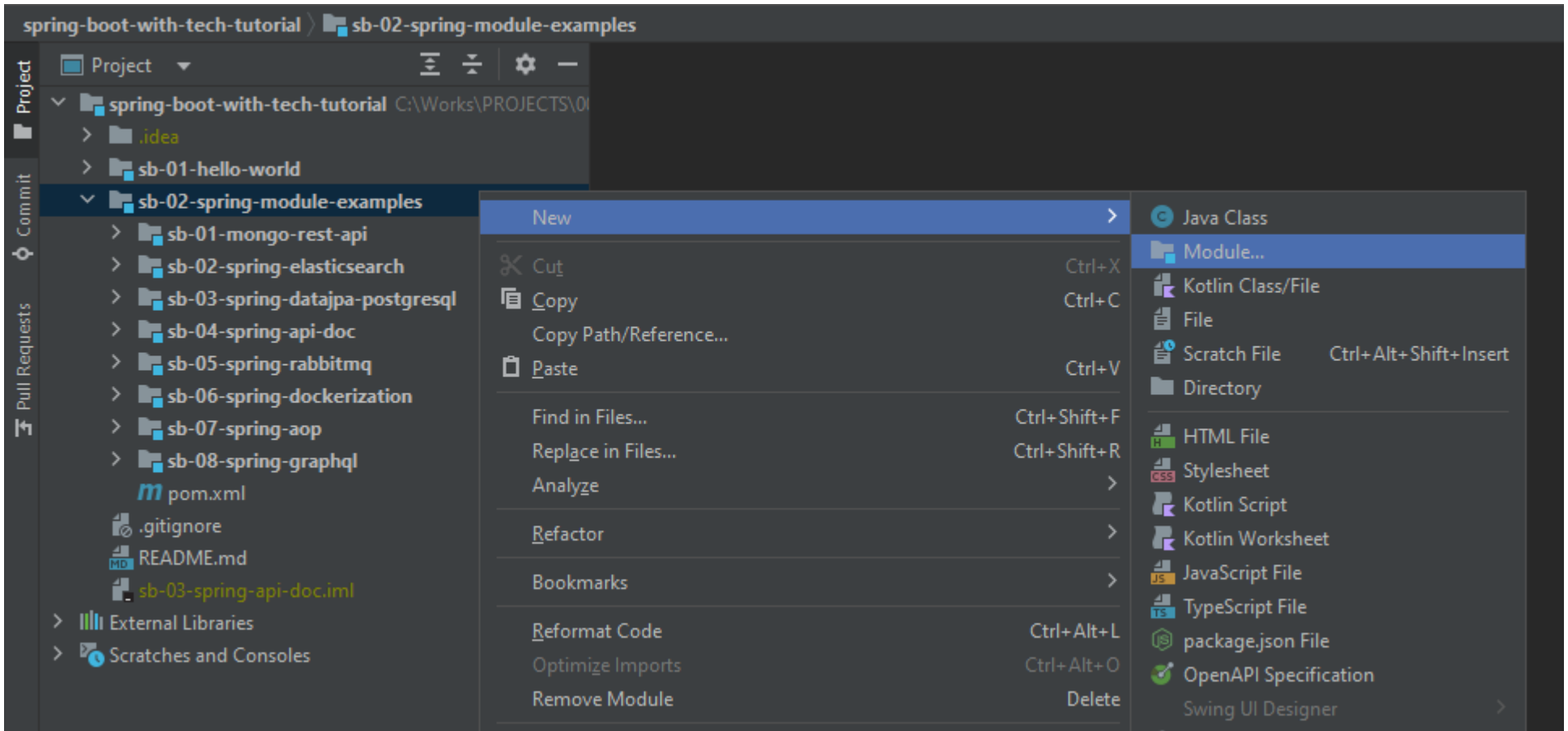
<https://redis.com/redis-enterprise/redis-insight/>







Spring Boot uygulamamızı oluşturalım.





Project pom.xml (sb-09-spring-redis-cache) x

```
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
5
6      <parent>
7          <groupId>org.springframework.boot</groupId>
8          <artifactId>spring-boot-starter-parent</artifactId>
9          <version>2.7.0</version>
10         <relativePath/> <!-- lookup parent from repository -->
11     </parent>
12
13     <modelVersion>4.0.0</modelVersion>
14
15     <artifactId>sb-09-spring-redis-cache</artifactId>
16
17     <properties>
18         <maven.compiler.source>11</maven.compiler.source>
19         <maven.compiler.target>11</maven.compiler.target>
20     </properties>
21
22     <dependencies>
23         <dependency>
24             <groupId>org.springframework.boot</groupId>
25             <artifactId>spring-boot-starter-web</artifactId>
26         </dependency>
27         <dependency>
28             <groupId>org.springframework.boot</groupId>
29             <artifactId>spring-boot-starter-data-redis</artifactId>
30         </dependency>
31         <dependency>
32             <groupId>redis.clients</groupId>
33             <artifactId>jedis</artifactId>
34         </dependency>
35     </dependencies>
36
37     <build>
38
39     <plugins>
```

spring-boot-with-tech-tutorial / sb-02-spring-module-examples / sb-09-spring-redis-cache / src / main / java / com / sb / gltkn / config / AppConfigConfiguration

```
1 package com.sb.gltkn.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
6 import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;
7 import org.springframework.data.redis.core.RedisTemplate;
8
9 @Configuration
10 public class AppConfigConfiguration {
11
12     1 usage
13     @Bean
14     public JedisConnectionFactory jedisConnectionFactory(){
15         RedisStandaloneConfiguration configuration = new RedisStandaloneConfiguration( hostName: "192.168.99.102", port: 6379);
16         return new JedisConnectionFactory(configuration);
17     }
18
19     @Bean
20     public RedisTemplate redisTemplate(){
21         RedisTemplate template = new RedisTemplate();
22         template.setConnectionFactory(jedisConnectionFactory());
23         return template;
24     }
25 }
```

```
1 package com.sb.gltkn.service;
2
3 import org.springframework.stereotype.Service;
4
5 2 usages
6 @Service
7 public class RedisCacheService {
8
9     1 usage
10    public String loadCachingOperation() throws InterruptedException {
11        Thread.sleep(5000L);
12        return "loadCachingOperation metodu tamamlandı.";
13    }
14 }
```

```
1 package com.sb.gltkn.controller;
2
3 import com.sb.gltkn.service.RedisCacheService;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.RestController;
8
9 @RestController
10 @RequestMapping("/redis")
11 public class RedisCacheController {
12
13     1 usage
14     @Autowired
15     private RedisCacheService redisCacheService;
16
17     @GetMapping
18     public String getRedisCacheOperation() throws InterruptedException {
19         return redisCacheService.loadCachingOperation();
20     }
21 }
```

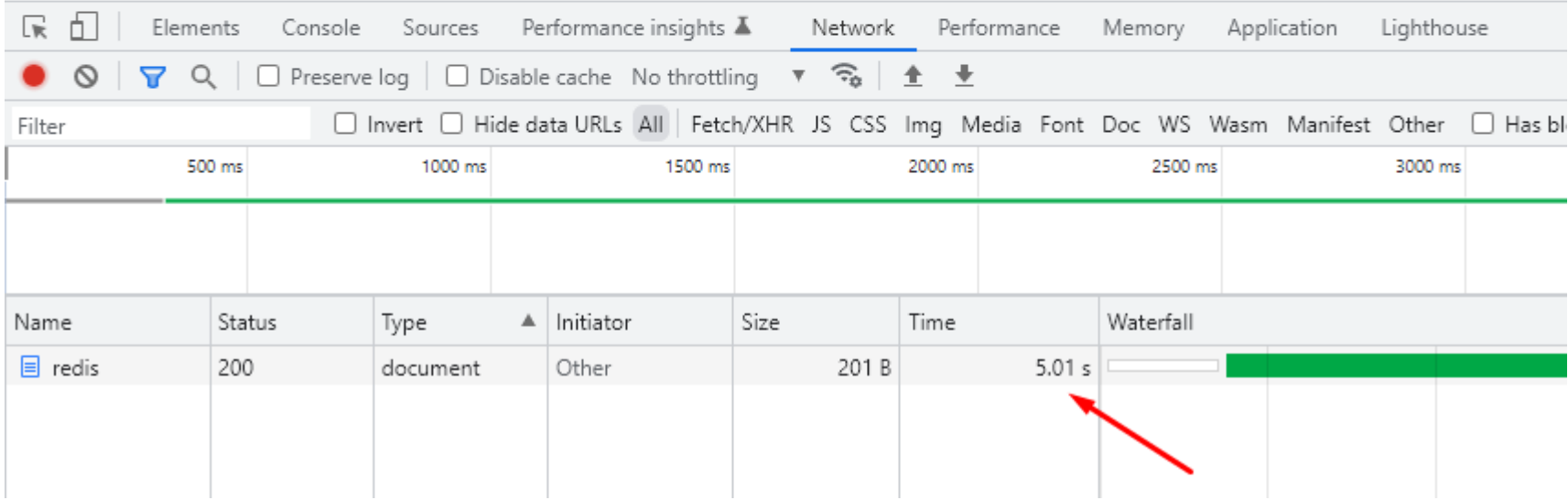
Uygulamamızı şimdi çalıştıralım.

<http://localhost:8080/redis>

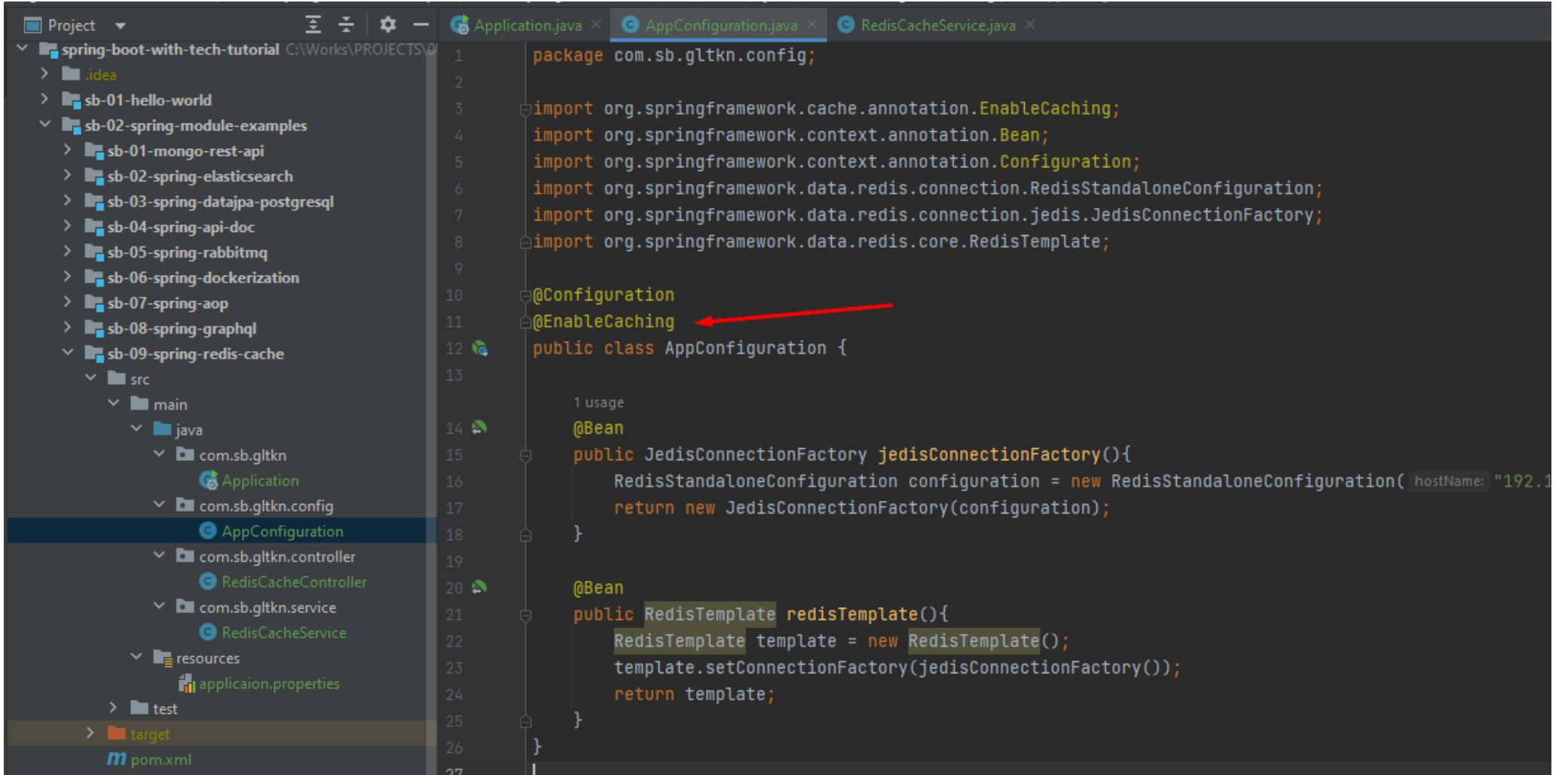
Get isteğinde bulunduğumuzda 5 saniye thread bekletti ve sonra response verdi.

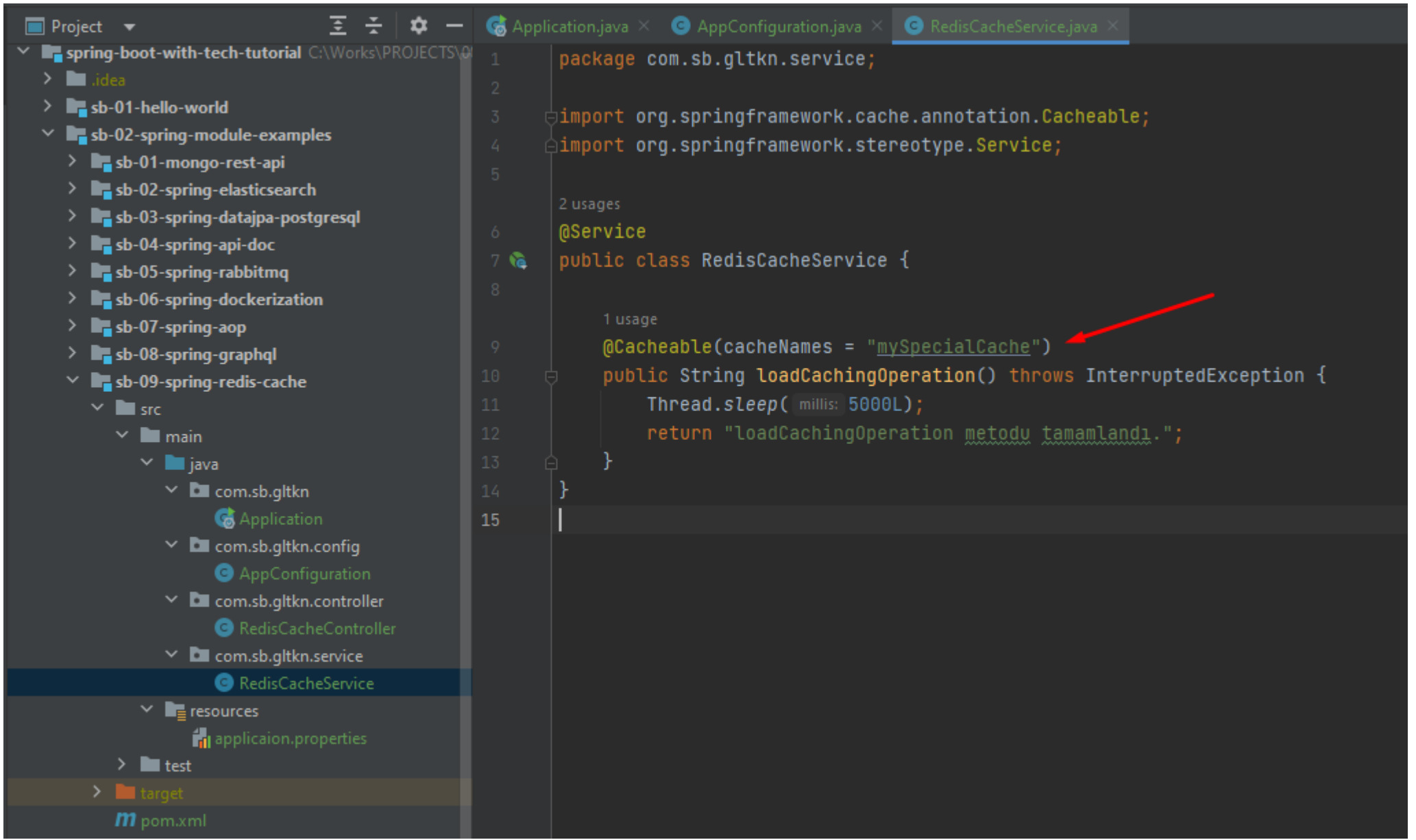


loadCachingOperation metodu tamamlandı.



Şimdi Caching uygulayalım.

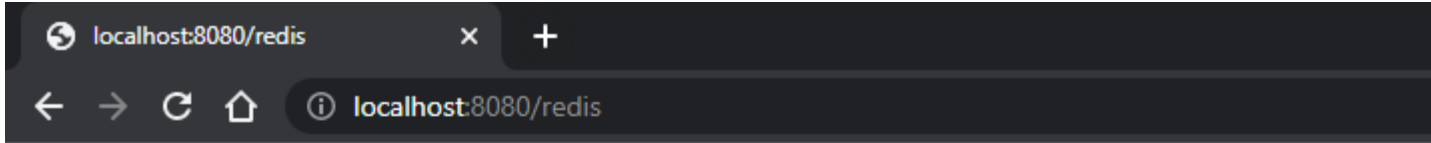




Uygulamamızı tekrar çalıştıralım.

<http://localhost:8080/redis>

Uygulama ayağı kalktı ve ilk isteğimizi atalım.  
İlk isteğimizde `loadCachingOperation` metodu çalıştı ve 5 saniye bekledi. Bu ilk istekte caching gerçekleşti. Sonraki isteklerde redis cache'den veri getirilecektir.



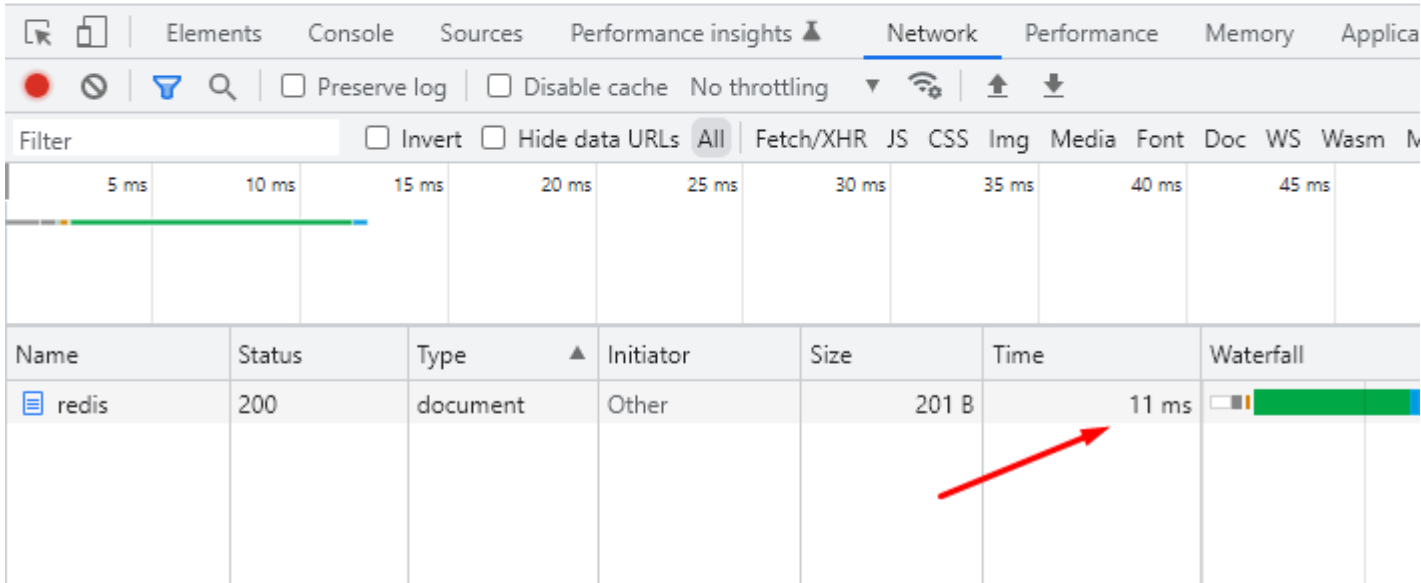
loadCachingOperation metodu tamamlandı.

Name	Status	Type	Initiator	Size	Time	Waterfall
redis	200	document	Other	201 B	5.22 s	

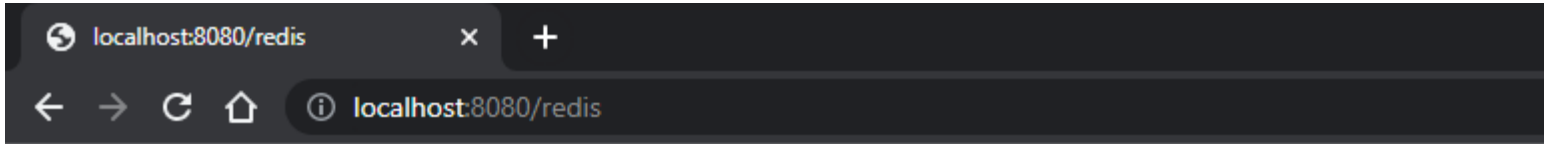
Görüldüğü üzere 2. isteğimizde cache'den aldığı için 11 milisaniye sürmüştür.



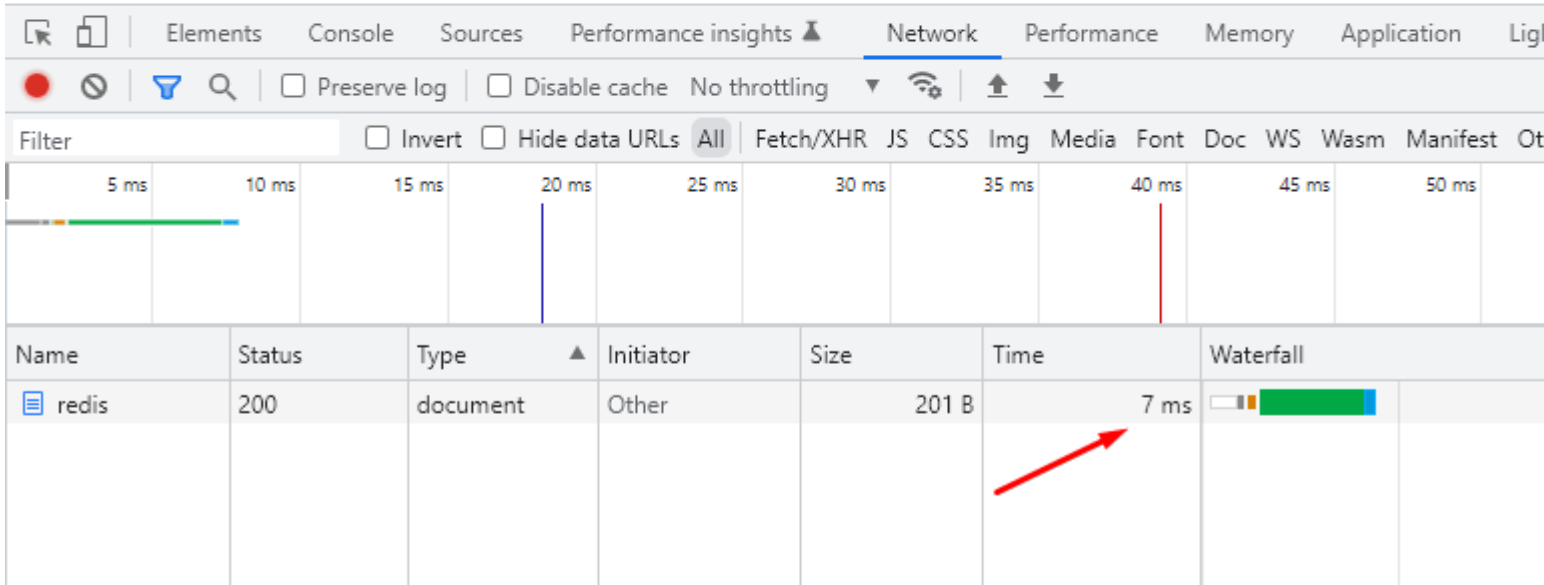
loadCachingOperation metodu tamamlandı.



Görüldüğü üzere 3. isteğimizde cache'den aldığı için 7 milisaniye sürmüştür.

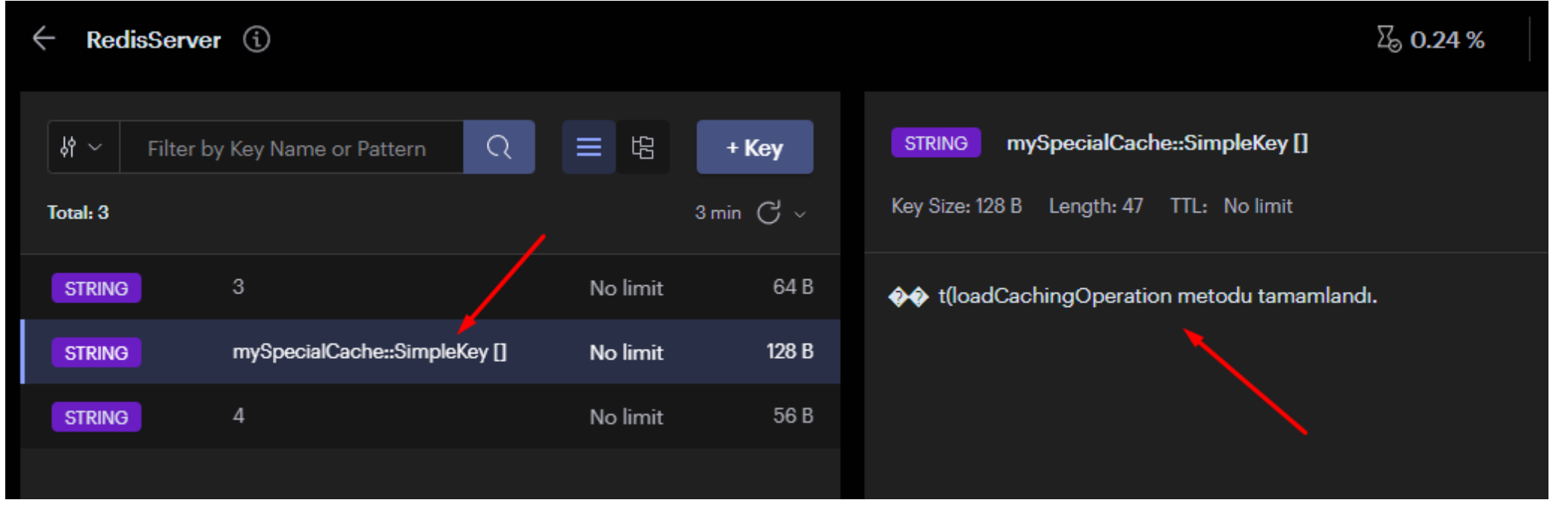


loadCachingOperation metodu tamamlandı.





Cache'lenmiş olduğunu aşağıdaki uygulama üzerinde görebiliriz.



The screenshot shows the RedisServer interface. On the left, a table lists keys:

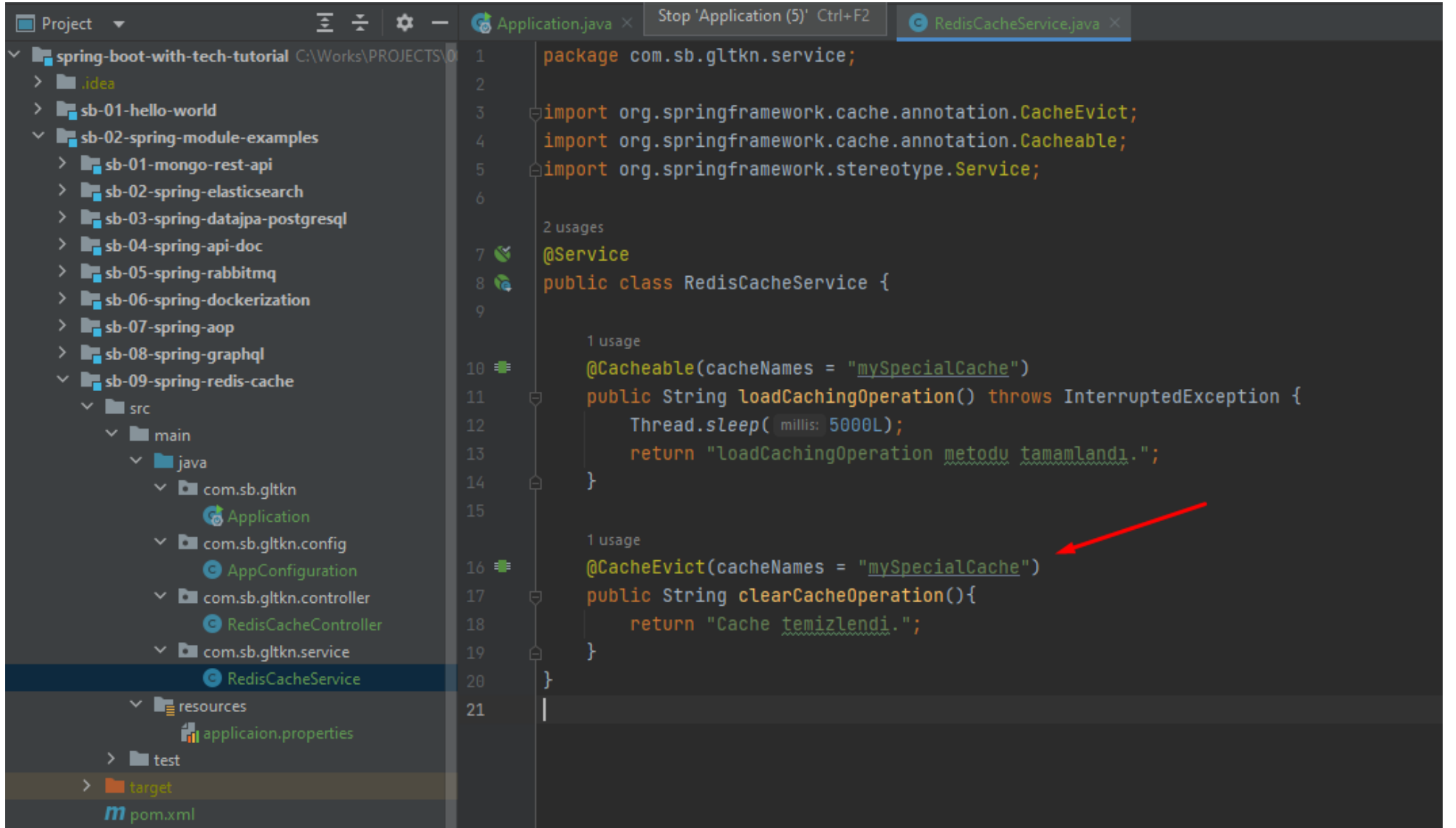
Type	Count	Key Name	TTL	Size
STRING	3		No limit	64 B
STRING	1	mySpecialCache::SimpleKey []	No limit	128 B
STRING	4		No limit	56 B

A red arrow points to the key 'mySpecialCache::SimpleKey []'. On the right, the detailed view of this key is shown:

- Type: STRING
- Key Name: mySpecialCache::SimpleKey []
- Key Size: 128 B
- Length: 47
- TTL: No limit
- Value: t(loadCachingOperation metodu tamamlandı.)

A red arrow points to the value 't(loadCachingOperation metodu tamamlandı.)'.

Cache temizleme metodunu yazalım.



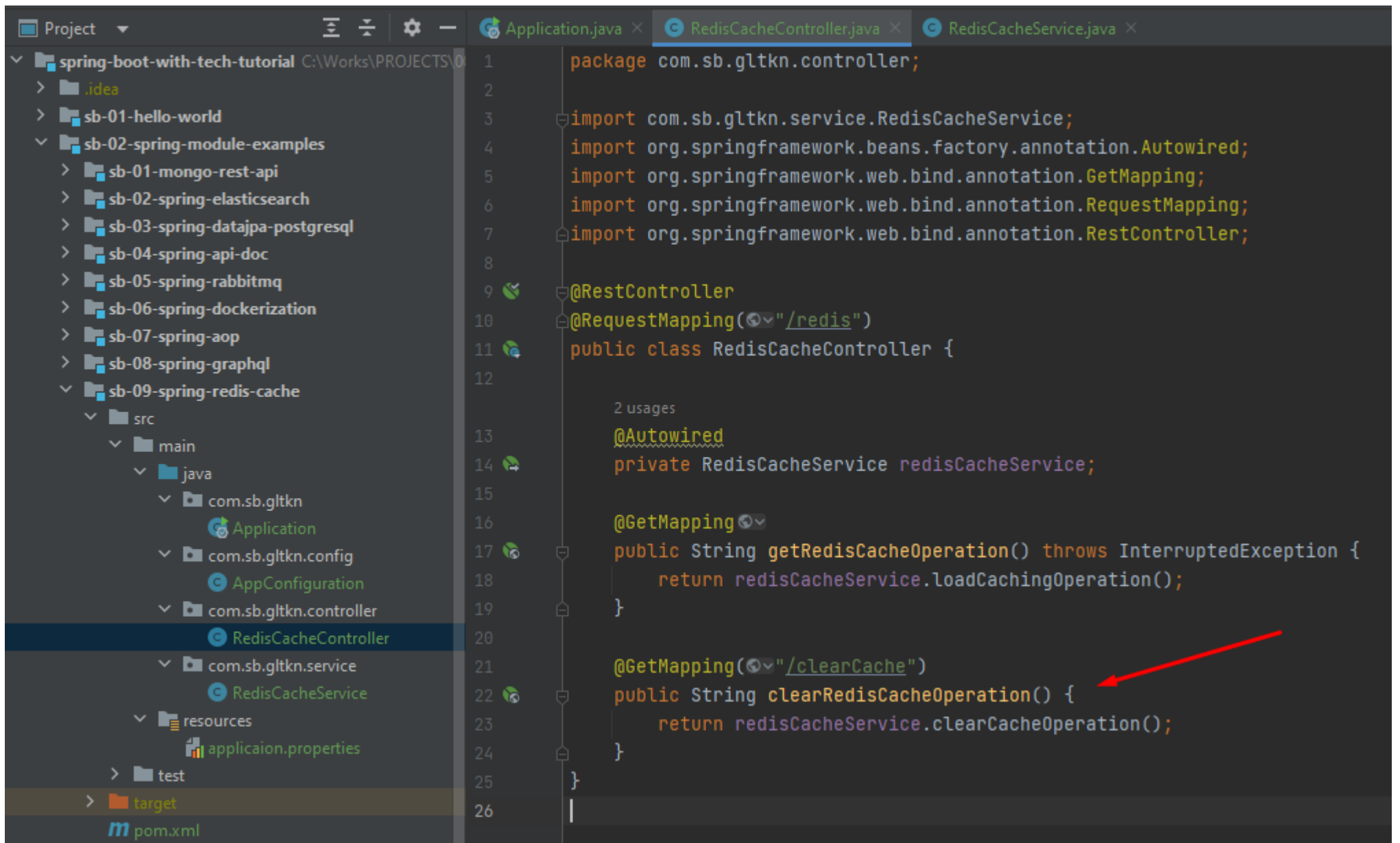
The screenshot shows an IDE with the following structure:

- Project: spring-boot-with-tech-tutorial
- sb-02-spring-module-examples
- sb-02-spring-elasticsearch
- sb-03-spring-datajpa-postgresql
- sb-04-spring-api-doc
- sb-05-spring-rabbitmq
- sb-06-spring-dockerization
- sb-07-spring-aop
- sb-08-spring-graphql
- sb-09-spring-redis-cache
  - src
    - main
      - java
        - com.sb.gltkn
          - Application
          - AppConfiguration
          - RedisCacheController
          - RedisCacheService

The code in RedisCacheService.java is as follows:

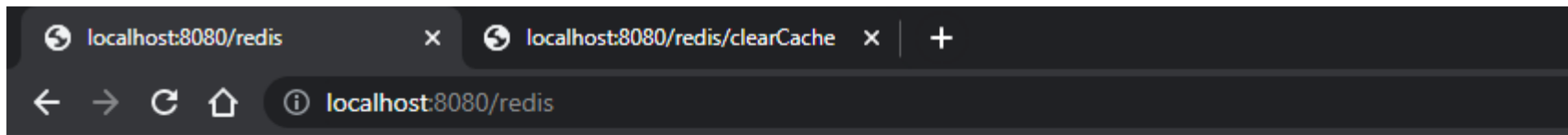
```
1 package com.sb.gltkn.service;
2
3 import org.springframework.cache.annotation.CacheEvict;
4 import org.springframework.cache.annotation.Cacheable;
5 import org.springframework.stereotype.Service;
6
7 @Service
8 public class RedisCacheService {
9
10     @Cacheable(cacheNames = "mySpecialCache")
11     public String loadCachingOperation() throws InterruptedException {
12         Thread.sleep( millis: 5000L);
13         return "loadCachingOperation metodu tamamlandı.";
14     }
15
16     @CacheEvict(cacheNames = "mySpecialCache")
17     public String clearCacheOperation(){
18         return "Cache temizlendi.";
19     }
20 }
21
```

A red arrow points to the @CacheEvict annotation on line 16.

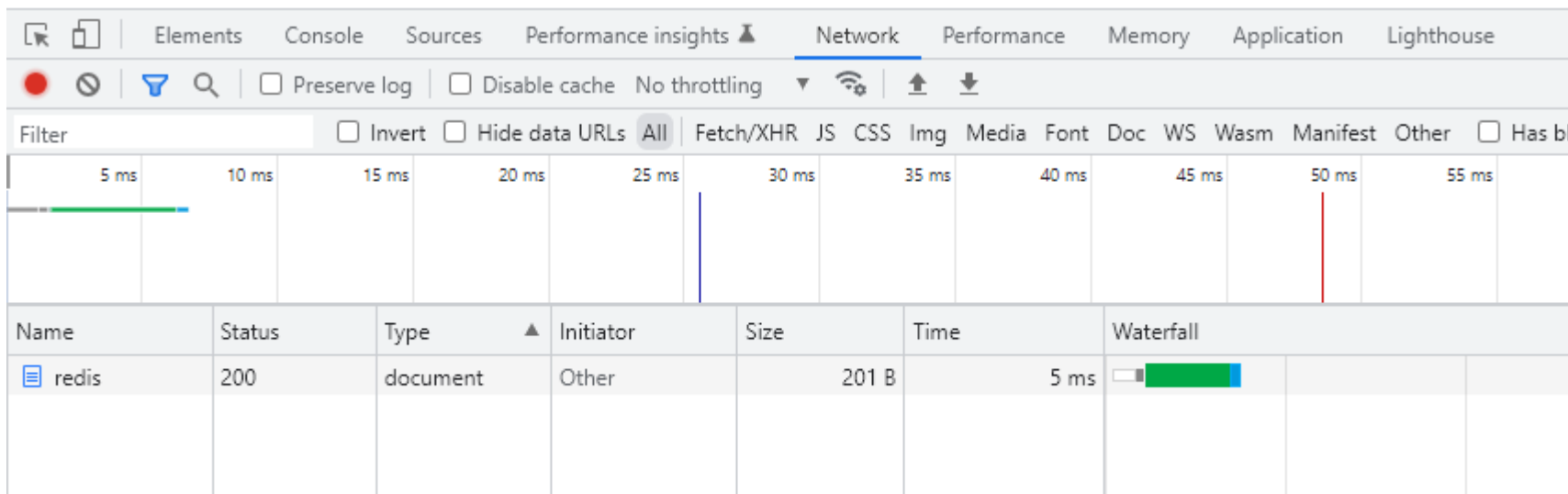


```
1 package com.sb.gltkn.controller;
2
3 import com.sb.gltkn.service.RedisCacheService;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.RestController;
8
9 @RestController
10 @RequestMapping("/redis")
11 public class RedisCacheController {
12
13     2 usages
14     @Autowired
15     private RedisCacheService redisCacheService;
16
17     @GetMapping
18     public String getRedisCacheOperation() throws InterruptedException {
19         return redisCacheService.loadCachingOperation();
20     }
21
22     @GetMapping("/clearCache")
23     public String clearRedisCacheOperation() {
24         return redisCacheService.clearCacheOperation();
25     }
26 }
```

<http://localhost:8080/redis>



loadCachingOperation metodu tamamlandı.

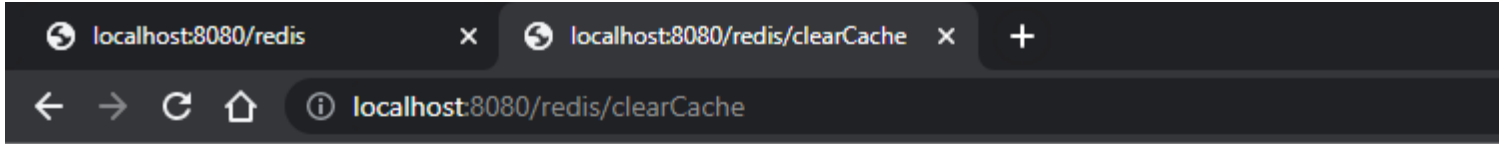


Name	Status	Type	Initiator	Size	Time	Waterfall
redis	200	document	Other	201 B	5 ms	

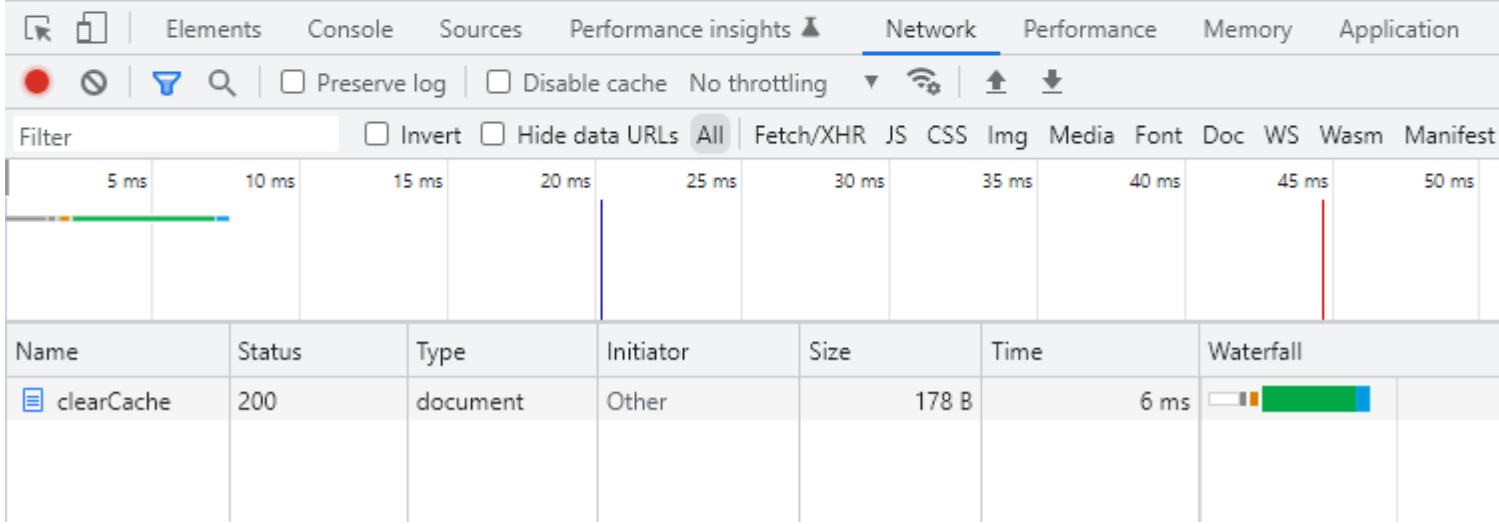


Şimdi cache'i temizleyelim.

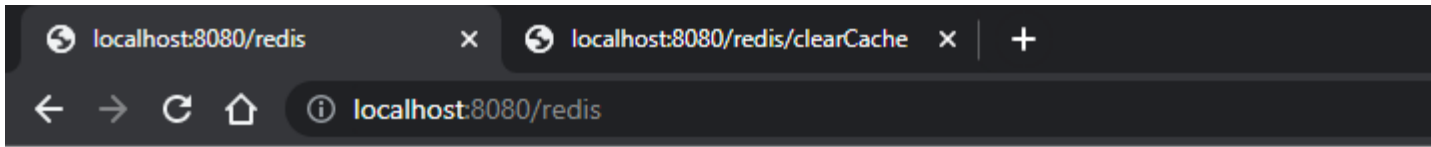
<http://localhost:8080/redis/clearCache>



Cache temizlendi.



Cache temizlendikten sonra istek attığımızda 5 saniye sürdü.



loadCachingOperation metodu tamamlandı.

