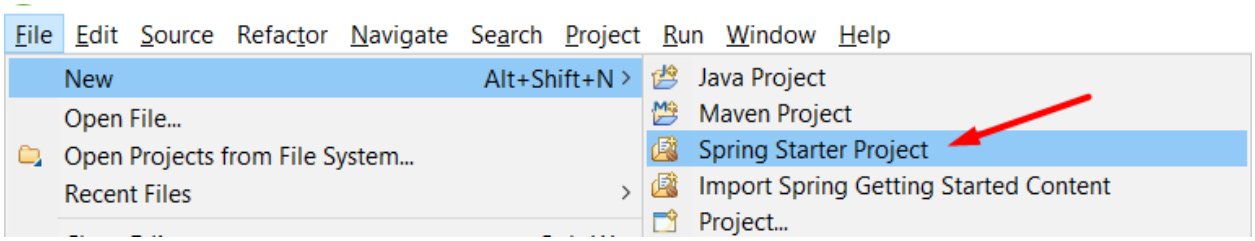
# PROYECTO “REDIS Producto”

## Paso 1: Crear el Proyecto en STS

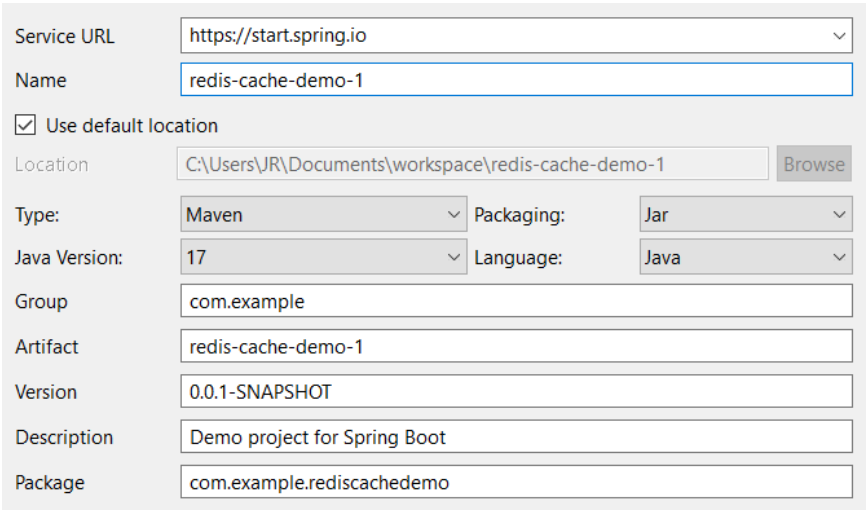
Abrir Spring Tool Suite (STS).

Ir a **File → New → Spring Starter Project**

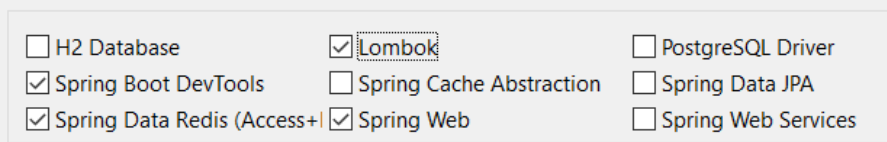



Completar los datos del proyecto:

- **Name:** redis-cache-demo
- **Type:** Maven
- **Java Version:** 17
- **Group:** com.example
- **Artifact:** redis-cache-demo
- **Package:** com.example.rediscachedemo
- **Packaging:** Jar




- **Dependencies:**
  - **Spring Web (Para la API REST).**
  - **Spring Data Redis (Access + Driver)** (Para conectarnos a Redis).
  - **Lombok** (Para reducir código boilerplate).
  - **Spring Boot DevTools** (Opcional, para recarga en vivo).




Hacer clic en **Finish** para generar el proyecto.

Luego en el archivo pom agregaremos la dependencia del **Spring Boot Starter Cache** (Para usar @Cacheable y @CacheEvict).

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

# Paso 2: Configurar Redis en application.properties

Añadimos la conexión a nuestra base de datos Redis.

```
spring.application.name=redis-cache-demo

# Configuracion de Redis
spring.data.redis.host=redis-15022.c82.us-east-1-2.ec2.redns.redis-cloud.com
spring.data.redis.port=15022
spring.data.redis.username=default
spring.data.redis.password=x4M4OjOpwpwr5B331n4NH43FSIRgrwkm

# Habilitar cache en Spring Boot
spring.cache.type=redis
```

Explicación:

- Se configuran los datos de conexión a Redis.
- **Se activa el uso de caché con spring.cache.type=redis** para que Spring Boot use Redis en @Cacheable.

# Paso 3: Crear la Clase Producto

Creamos la clase Producto

```java
package com.example.rediscachedemo.model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import java.io.Serializable;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Producto implements Serializable {
    private Long id;
    private String nombre;
    private double precio;
}
```

Explicación:

- Producto representa los datos de un producto.
- Serializable es necesario porque los datos en Redis **se serializan antes de almacenarse**.
- **Usamos Lombok (@Getter, @Setter, etc.)** para evitar escribir código innecesario.

# Paso 4: Crear el ProductoService con @Cacheable

Creamos la lógica de negocio en el service ProductoService

```java
package com.example.rediscachedemo.service;

import com.example.rediscachedemo.model.Producto;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;
import java.util.HashMap;
import java.util.Map;

@Service
public class ProductoService {
    private final Map<Long, Producto> bdSimulada = new HashMap<>();

    public ProductoService() {
        // Simulando productos en "base de datos"
        bdSimulada.put(1L, new Producto(1L, "Laptop", 1200.00));
        bdSimulada.put(2L, new Producto(2L, "Smartphone", 800.00));
        bdSimulada.put(3L, new Producto(3L, "Tablet", 500.00));
    }

    @Cacheable(value = "productos", key = "#id")
    public Producto obtenerProducto(Long id) {
        System.out.println("Buscando producto en la base de datos...");
        return bdSimulada.get(id);
    }

    @CacheEvict(value = "productos", key = "#id")
    public void eliminarProductoDelCache(Long id) {
        System.out.println("Producto eliminado de la caché con ID: " + id);
    }
}
```

Explicación:

- Simulamos una BD **con HashMap<Long, Producto>.**

**@Cacheable(value = "productos", key = "#id")**:

- Si el producto **NO está en Redis**, lo busca en la BD y lo guarda en la caché.
- Si el producto **YA está en Redis**, lo obtiene en **milisegundos** sin acceder a la BD.

**@CacheEvict(value = "productos", key = "#id")**:

- Elimina el producto de la caché cuando se actualiza o se quiere invalidar.

# Paso 5: Crear el ProductoController

Creamos el controller ProductoController

```java
package com.example.rediscachedemo.controller;

import com.example.rediscachedemo.model.Producto;
import com.example.rediscachedemo.service.ProductoService;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/productos")
public class ProductoController {
    private final ProductoService productoService;

    public ProductoController(ProductoService productoService) {
        this.productoService = productoService;
    }

    @GetMapping("/{id}")
    public Producto obtenerProducto(@PathVariable Long id) {
        return productoService.obtenerProducto(id);
    }

    @DeleteMapping("/{id}/cache")
    public String eliminarProductoDelCache(@PathVariable Long id) {
        productoService.eliminarProductoDelCache(id);
        return "Producto eliminado del caché.";
    }
}
```

Explicación:

**GET /productos/{id}**:

- **Primera vez**: Busca en la BD y lo guarda en Redis.
- **Siguientes veces**: Obtiene de Redis **sin acceder a la BD**.

**DELETE /productos/{id}/cache**:

- Borra el producto **de la caché** (por ejemplo, si el precio cambia).

# Paso 6: Habilitar la Caché en la Aplicación

Agregamos en la clase principal de nuestra aplicación la anotación **@EnableCaching**, lo cual habilita el soporte de caché en Spring Boot.

```java
package com.example.rediscachedemo;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@EnableCaching
public class RedisCacheDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(RedisCacheDemoApplication.class, args);
    }

}
```

Explicación:

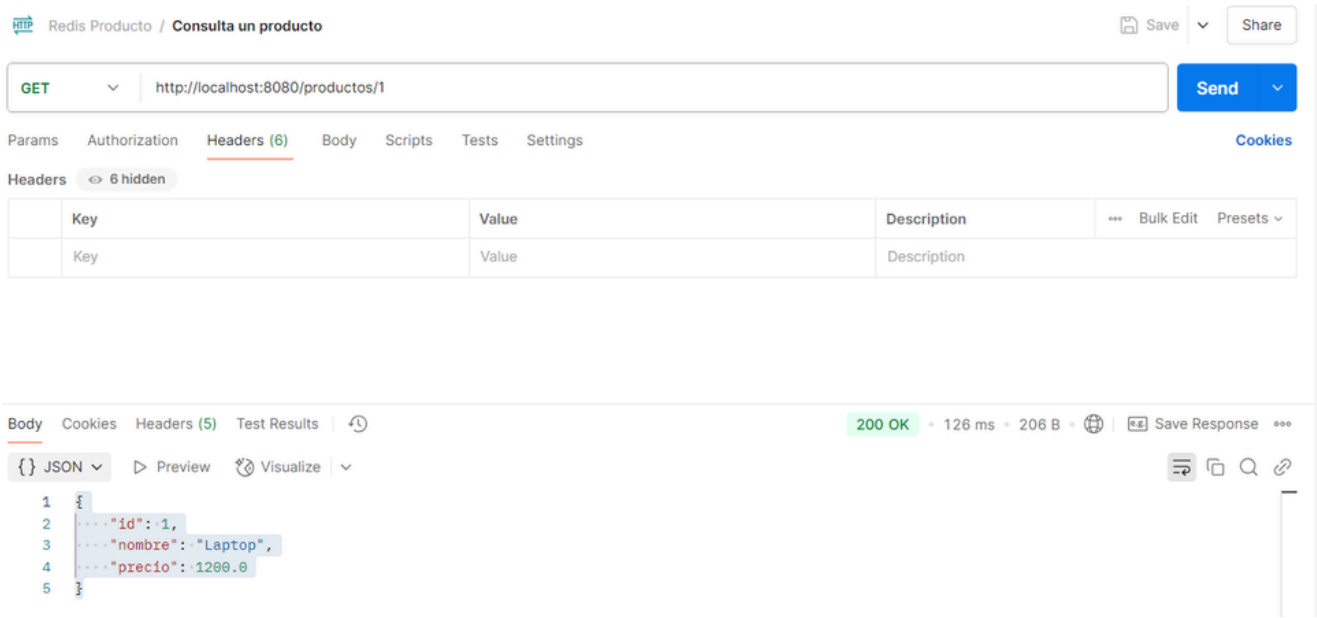
- **@EnableCaching** es obligatorio para que **@Cacheable** funcione**.**

# Paso 7: Ejecutar y Probar

- **GET** http://localhost:8080/productos/1

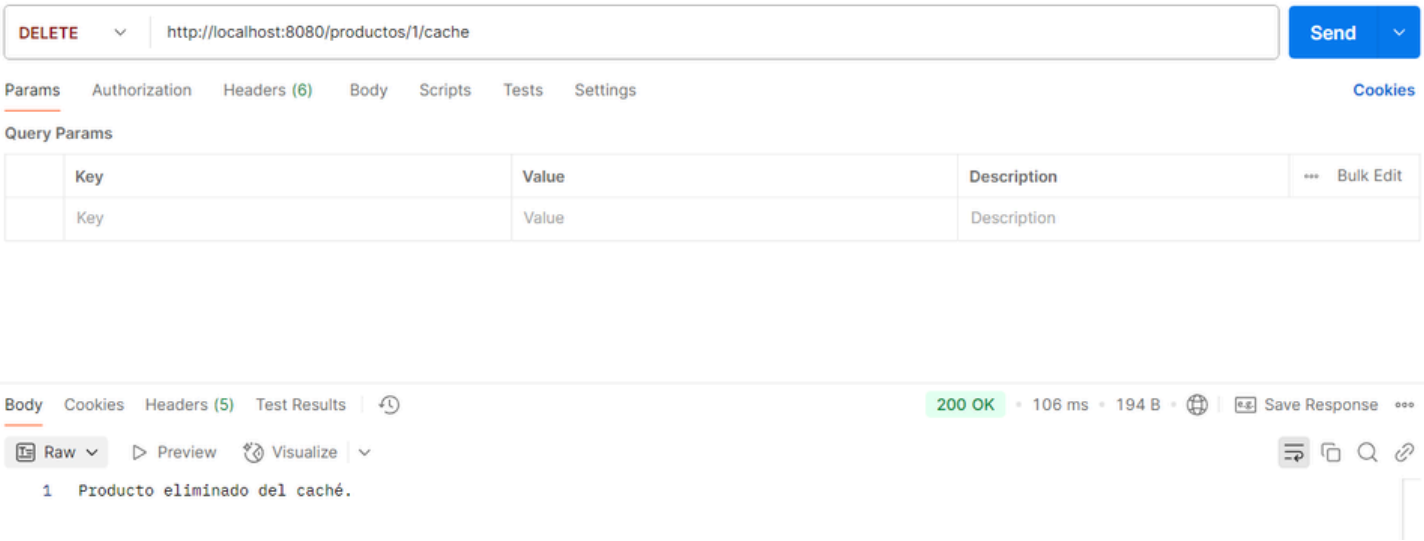
Respuesta esperada

```json
{
  "id": 1,
  "nombre": "Laptop",
  "precio": 1200.0
}
```




- **DELETE** http://localhost:8080/productos/1/cache

Respuesta esperada


DELETE
http://localhost:8080/productos/1/cache
Send
Params
Authorization
Headers (6)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description


Body
Cookies
Headers (5)
Test Results
200 OK
106 ms
194 B
Save Response
Raw
Preview
Visualize
1 Producto eliminado del caché.