Spring Boot Redis example

spring boot with redis example code from scratch, give complete code in step by step

here use redis as backend database

Here's a **complete working Spring Boot + Redis** CRUD example **from scratch**, using **Redis as the backend database** (no relational DB). We'll store simple `User` records in Redis as hashes using **Spring Data Redis**.

✅ Stack:

- Spring Boot 3+
- Spring Data Redis
- Embedded Redis (for local dev, optional)
- Redis running on port 6379

✅ Goal:

CRUD operations using Redis as the **primary database** (no other DB).

📁 Project Structure:

```
css

spring-boot-redis-example/
├── src/
│   └── main/
│       ├── java/com/example/redis/
│       │   ├── controller/
│       │   │   └── UserController.java
│       │   ├── model/
│       │   │   └── User.java
│       │   ├── repository/
│       │   │   └── UserRepository.java
```

```
│       │   ├── RedisExampleApplication.java
│       └── resources/
│           └── application.yml
└── pom.xml
```

---

① `pom.xml`

xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>spring-boot-redis-example</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring-boot-redis-example</name>
    <description>Spring Boot + Redis (No DB)</description>

    <properties>
        <java.version>17</java.version>
        <spring-boot.version>3.2.4</spring-boot.version>
    </properties>

    <dependencies>
        <!-- Spring Boot Web -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Spring Boot Data Redis -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>

        <!-- Embedded Redis (for local testing, optional) -->
```

```
        <dependency>
            <groupId>it.ozimov</groupId>
            <artifactId>embedded-redis</artifactId>
            <version>0.7.3</version>
        </dependency>

        <!-- Lombok -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Spring Boot Plugin -->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

---

② `application.yml`

yaml

```yaml
spring:
  data:
    redis:
      host: localhost
      port: 6379
  application:
    name: redis-example

server:
  port: 8080
```

---

③ `User.java` (Model)

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

import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.redis.core.RedisHash;

import java.io.Serializable;

@RedisHash("User")  // Stored as Redis hash with key prefix "User"
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User implements Serializable {
    @Id
    private String id;
    private String name;
    private String email;
}
```

---

④ `UserRepository.java`

```java
package com.example.redis.repository;

import com.example.redis.model.User;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends CrudRepository<User, String> {
}
```

---

⑤ `UserController.java`

java

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

import com.example.redis.model.User;
import com.example.redis.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Optional;

@RestController
@RequestMapping("/users")
@RequiredArgsConstructor
public class UserController {

    private final UserRepository userRepository;

    @PostMapping
    public ResponseEntity<User> save(@RequestBody User user) {
        User saved = userRepository.save(user);
        return ResponseEntity.ok(saved);
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> findById(@PathVariable String id) {
        Optional<User> user = userRepository.findById(id);
        return user.map(ResponseEntity::ok).orElse(ResponseEntity.notFound().build());
    }

    @GetMapping
    public Iterable<User> findAll() {
        return userRepository.findAll();
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> delete(@PathVariable String id) {
        userRepository.deleteById(id);
        return ResponseEntity.noContent().build();
    }
}
```

⑥ `RedisExampleApplication.java`

```java
package com.example.redis;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class RedisExampleApplication {

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

---

⑦ (Optional) Embedded Redis Config – for local development only

If you don't have Redis installed locally, use this:

```java
package com.example.redis;

import jakarta.annotation.PreDestroy;
import org.springframework.context.annotation.Configuration;
import redis.embedded.RedisServer;

import jakarta.annotation.PostConstruct;

@Configuration
public class EmbeddedRedisConfig {

    private RedisServer redisServer = new RedisServer(6379);

    @PostConstruct
    public void startRedis() {
        redisServer.start();
    }
```

```
    @PreDestroy
    public void stopRedis() {
        redisServer.stop();
    }
}
```

🧪 Test Using Postman or Curl

▶ Create User

bash

```
POST http://localhost:8080/users
Content-Type: application/json

{
  "id": "u123",
  "name": "Viswanath",
  "email": "viswa@example.com"
}
```

▶ Get User by ID

bash

```
GET http://localhost:8080/users/u123
```

▶ Get All Users

bash

```
GET http://localhost:8080/users
```

▶ Delete User

bash

```
DELETE http://localhost:8080/users/u123
```

✅ Redis Structure

This uses Redis `HASH` internally like:

```bash
HSET User:u123 name "Viswanath"
HSET User:u123 email "viswa@example.com"
```

✅ Run the App

If Redis is installed:

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

Then run:

```bash
mvn spring-boot:run
```

Would you like this project zipped or deployed on GitHub?

why added spring-boot-starter-data-redis can't we use data-jpa

Great question. Here's the **difference between** `spring-boot-starter-data-redis` **and** `spring-boot-starter-data-jpa`, and **why we used Redis-specific starter** in this case:

🔧 TL;DR:

Feature	`spring-boot-starter-data-jpa`	`spring-boot-starter-data-redis`
Purpose	For relational DBs like MySQL, Postgres	For NoSQL DB like Redis
Backend storage	Tables, Rows, SQL	Key-Value store, Hashes, Sets, etc.
ORM layer	Hibernate/JPA	RedisTemplate / Spring Data Redis
Supports Redis?	❌ No	✅ Yes
Supports SQL?	✅ Yes	❌ No

✅ Why we use `spring-boot-starter-data-redis` here?

You're building an app that uses **Redis as the backend database** — not MySQL, not PostgreSQL. So:

- `spring-boot-starter-data-redis` gives:
  - Redis-specific repository support (`RedisRepository` , `CrudRepository`)
  - Serialization support (objects → Redis data types)
  - Auto-configuration for Redis clients (Lettuce/Jedis)

```xml

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

This starter configures:

- Redis connection
- `RedisTemplate` or auto-maps model classes with `@RedisHash`
- CRUD over Redis

🚫 Why not use `spring-boot-starter-data-jpa`?

`spring-boot-starter-data-jpa` is designed **only for SQL databases**.

If you try to use it with Redis:

- It won't work — JPA expects a relational schema, entities, tables, and SQL queries.
- Redis is key-value, not relational — no schema, no foreign keys, no joins.

✅ When to use both?

If you have:

- MySQL/Postgres for primary data (via JPA)
- Redis for caching (via Redis)

Then, you can use **both starters** side by side:

```xml
<dependency> <!-- For DB -->
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency> <!-- For Redis cache -->
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

🧠 Summary:

Use case	Starter needed
Only Redis (as DB)	✅ `spring-boot-starter-data-redis`
Only SQL DB (MySQL/Postgres)	✅ `spring-boot-starter-data-jpa`

Use case	Starter needed
Redis for cache + SQL DB for storage	✅ Both

Let me know if you want an example with **both JPA + Redis cache**, or **Redis as cache layer** only.

give example of use RedisTemplate

Absolutely! Here's a **complete example of using** `RedisTemplate` in Spring Boot to perform custom Redis operations like setting, getting, and deleting keys — independent of Spring Data repositories.

✅ Why Use `RedisTemplate`?

- `RedisTemplate<K, V>` gives you **low-level control** over Redis.
- You can:
  - Use different data types (`String`, `Hash`, `List`, `Set`, `ZSet`)
  - Customize serialization
  - Manually interact with keys and values

✅ Example: Spring Boot + `RedisTemplate` for Key-Value Ops

We'll create an API to manage simple `Product` data in Redis using `RedisTemplate`.

📁 Project Structure

```
arduino

src/
 ├── controller/
 │    └── ProductController.java
 ├── config/
 │    └── RedisConfig.java
```

```
├── model/
│   └── Product.java
├── service/
│   └── ProductService.java
└── RedisTemplateExampleApplication.java
```

---

① `pom.xml` – Required Dependency

xml

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

---

② `Product.java`

java

```java
package com.example.model;

import lombok.*;

import java.io.Serializable;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Product implements Serializable {
  private String id;
  private String name;
  private double price;
}
```

---

③ `RedisConfig.java` – Configure `RedisTemplate`

java

```java
package com.example.config;

import com.example.model.Product;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.serializer.*;
import org.springframework.data.redis.core.RedisTemplate;

@Configuration
public class RedisConfig {

    @Bean
    public RedisTemplate<String, Product> redisTemplate(RedisConnectionFactory factory) {
        RedisTemplate<String, Product> template = new RedisTemplate<>();
        template.setConnectionFactory(factory);

        // Use String keys and JSON-serialized Product values
        template.setKeySerializer(new StringRedisSerializer());
        template.setValueSerializer(new GenericJackson2JsonRedisSerializer());

        return template;
    }
}
```

---

④ `ProductService.java`

```java
package com.example.service;

import com.example.model.Product;
import lombok.RequiredArgsConstructor;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

import java.util.concurrent.TimeUnit;

@Service
@RequiredArgsConstructor
```

```
public class ProductService {

    private final RedisTemplate<String, Product> redisTemplate;
    private static final String PREFIX = "Product:";

    public void saveProduct(Product product) {
        redisTemplate.opsForValue().set(PREFIX + product.getId(), product);
        // Optional: set expiration
        redisTemplate.expire(PREFIX + product.getId(), 1, TimeUnit.HOURS);
    }

    public Product getProductById(String id) {
        return redisTemplate.opsForValue().get(PREFIX + id);
    }

    public void deleteProduct(String id) {
        redisTemplate.delete(PREFIX + id);
    }

    public boolean exists(String id) {
        return Boolean.TRUE.equals(redisTemplate.hasKey(PREFIX + id));
    }
}
```

---

⑤ `ProductController.java`

java

```
package com.example.controller;

import com.example.model.Product;
import com.example.service.ProductService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/products")
@RequiredArgsConstructor
public class ProductController {
```

```
    private final ProductService productService;

    @PostMapping
    public ResponseEntity<String> save(@RequestBody Product product) {
        productService.saveProduct(product);
        return ResponseEntity.ok("Product saved to Redis");
    }

    @GetMapping("/{id}")
    public ResponseEntity<?> get(@PathVariable String id) {
        Product product = productService.getProductById(id);
        return product != null ? ResponseEntity.ok(product) : ResponseEntity.notFound().build();
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<String> delete(@PathVariable String id) {
        productService.deleteProduct(id);
        return ResponseEntity.ok("Deleted");
    }

    @GetMapping("/{id}/exists")
    public ResponseEntity<Boolean> exists(@PathVariable String id) {
        return ResponseEntity.ok(productService.exists(id));
    }
}
```

---

⑥ `RedisTemplateExampleApplication.java`

java

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class RedisTemplateExampleApplication {
    public static void main(String[] args) {
        SpringApplication.run(RedisTemplateExampleApplication.class, args);
```

```
  }
}
```

🧪 Sample API Calls

```http
POST /products
{
  "id": "p101",
  "name": "Laptop",
  "price": 74999.0
}
```

```http
GET /products/p101
```

```http
DELETE /products/p101
```

```http
GET /products/p101/exists
```

✅ Summary of `RedisTemplate` Usage

Operation	Method Used
Set value	`opsForValue().set(key, value)`
Get value	`opsForValue().get(key)`
Delete key	`delete(key)`
Check existence	`hasKey(key)`

Operation	Method Used
Expiration	`expire(key, time, unit)`

Would you like examples of:

- `RedisTemplate` with `Hash`?
- `List`, `Set`, or `SortedSet` Redis types?

Let me know and I'll extend it.