Project Documentation: Spring Boot Product Management Application

Application Overview

This document provides detailed information about the Spring Boot-based Product Management Application. The project includes functionalities to create, retrieve, update, and delete product information, along with caching, asynchronous processing, and retry mechanisms.

Project Set

1. Set up the MySQL database and update connection details in application.properties.
2. Configure Redis for caching.
3. Import the project into your IDE.
4. Run the ProjectApplication class.
5. Access the API endpoints through tools like Postman or a web browser.

1. Project Configuration

1.1 Dependencies

- Spring Boot Starter Web
- Spring Boot Starter Data JPA
- Spring Boot Starter Cache
- MySQL Driver
- Redis
- Spring Retry
- Spring Boot Starter Validation

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
	<parent>
		<groupId>org.springframework.boot</groupId>
```

```xml
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>3.4.1</version>
		<relativePath /> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.demo</groupId>
	<artifactId>project</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>project</name>
	<description>Demo project for Spring Boot application</description>
	<url />
	<licenses>
		<license />
	</licenses>
	<developers>
		<developer />
	</developers>
	<scm>
		<connection />
		<developerConnection />
		<tag />
		<url />
	</scm>
	<properties>
		<java.version>17</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-redis</artifactId>
		</dependency>
		<dependency>
			<groupId>redis.clients</groupId>
			<artifactId>jedis</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
```

```
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.retry</groupId>
            <artifactId>spring-retry</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

1.2 Application Properties

The application is configured using the following properties:

```
spring.application.name=project


server.port=8088

spring.datasource.url=jdbc:mysql://localhost:3306/product
spring.datasource.username=root
spring.datasource.password=root123
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update

spring.cache.type=redis
spring.cache.redis.cache-null-values=true

server.ssl.key-alias=projectProduct
server.ssl.key-store=classpath:projectproduct.jks
server.ssl.key-store-password=project
server.ssl.key-store-type=JKS
```

2. Code Overview

2.1 Main Application Class

The ProjectApplication class serves as the entry point for the application. It enables caching, asynchronous processing, and retry mechanisms.

```
package com.demo.project.config;

import java.util.concurrent.Executor;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

@Configuration
public class AsyncConfiguration{

	@Bean(name = "taskExecutor")
	 Executor taskExecutor()
	{
		ThreadPoolTaskExecutor executor =  new
ThreadPoolTaskExecutor();
		executor.setCorePoolSize(5);
		executor.setMaxPoolSize(10);
		executor.setQueueCapacity(25);
		executor.setThreadNamePrefix("AsyncExecutor-");
		return executor;
	}

}
```

2.2 Entity Class

The Product entity represents the data model for the application.

```
package com.demo.project.entity;

import java.io.Serializable;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Product implements Serializable{

	/**
```

```
	 *
	 */
	private static final long serialVersionUID = 1L;
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;
	private String productName;
	private int price;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getProductName() {
		return productName;
	}
	public void setProductName(String productName) {
		this.productName = productName;
	}
	public int getPrice() {
		return price;
	}
	public void setPrice(int price) {
		this.price = price;
	}
	@Override
	public String toString() {
		return "Product [id=" + id + ", productName=" + productName
+ ", price=" + price + "]";
	}

}
```

2.3 Repository Interface

The ProductRepository interface extends JpaRepository for data access operations.

```
package com.demo.project.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import com.demo.project.entity.Product;

public interface ProductRepository extends JpaRepository<Product,
Integer> {

	List<Product> findByProductName(String productName);

	List<Product> findByPrice(int price);

}
```

2.4 Service Layer

The ProductService class contains business logic and implements retry, caching, and asynchronous processing.

```java
package com.demo.project.service;

import java.util.List;
import java.util.concurrent.CompletableFuture;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.cache.annotation.Caching;
import org.springframework.retry.annotation.Retryable;
import org.springframework.retry.annotation.Backoff;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;
import com.demo.project.entity.Product;
import com.demo.project.repository.ProductRepository;

@Service
public class ProductService {

	@Autowired
	private ProductRepository productRepository;

	 @Retryable(
				retryFor = { Exception.class },
				maxAttempts = 3,
				backoff = @Backoff(delay = 2000)
			)
	public void create(Product product)
	{
		productRepository.save(product);
	}

	@Async
	@Cacheable(value = "product")
	 @Retryable(
				retryFor = { Exception.class },
				maxAttempts = 3,
				backoff = @Backoff(delay = 2000)
			)
	public CompletableFuture<List<Product>> findAll()
	{
		return 
CompletableFuture.completedFuture(productRepository.findAll());
	}

	@Async
	@Cacheable(value = "product", key = "#id")
	 @Retryable(
				retryFor = { Exception.class },
				maxAttempts = 3,
				backoff = @Backoff(delay = 2000)
			)
	public CompletableFuture<Product> getById(int id)
	{
		return
CompletableFuture.completedFuture(productRepository.findById(id).orElse
Throw(()-> new ProductNotFoundException("No Data Found")));
	}

	public String update(Product product)
	{
		productRepository.save(product);
		return "Updated Successfully..!";
	}

	@Caching(
			evict = {@CacheEvict(value = "product", allEntries =
true),@CacheEvict(value = "product", key = "id")
			})
	public String delete(int id)
	{
		productRepository.deleteById(id);
		return "Deleted Successfully";
	}

	@Async
	@Cacheable(value = "product", key = "#productName")
	public CompletableFuture<List<Product>> findByProductName(String
productName)
	{
		//return
CompletableFuture.completedFuture(productRepository.findByProductName(p
roductName));
		 List<Product> products =
productRepository.findByProductName(productName);

		        if (products.isEmpty()) {
		            throw new ProductNotFoundException("No products
found with name: " + productName);
		        }

		        return CompletableFuture.completedFuture(products);
	}

	@Async
	@Cacheable(value = "product", key = "#price")
	public CompletableFuture<List<Product>> findByPrice(int price)
	{
		 List<Product> products =
productRepository.findByPrice(price);

		        if (products.isEmpty()) {
		            throw new ProductNotFoundException("No products
found with name: " + price);
		        }

		        return CompletableFuture.completedFuture(products);
	}

}
```

2.5 Controller Layer

The ProductController provides REST endpoints for client interaction.

```java
package com.demo.project.controller;

import java.util.List;
import java.util.concurrent.CompletableFuture;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.demo.project.entity.Product;
import com.demo.project.service.ProductNotFoundException;
import com.demo.project.service.ProductService;

@RestController
@RequestMapping("/api")
public class ProductController {

	@Autowired
	private ProductService productService;

	@PostMapping("/product")
	public String create(@RequestBody Product product) {
		productService.create(product);
		return "Successfully Posted Data..";
	}

	@GetMapping("/product")
	public CompletableFuture<List<Product>> getAll() {
		return productService.findAll();
	}

	@GetMapping("/product/{id}")
	public CompletableFuture<Product> getById(@PathVariable int id) {
		return productService.getById(id);
	}

	@GetMapping("/products/{productName}")
	public CompletableFuture<List<Product>>
getByProductName(@PathVariable String productName) {
		return productService.findByProductName(productName);
	}

	@GetMapping("/{price}")
	public CompletableFuture<List<Product>> getByPrice(@PathVariable
int price) {
```

```
            // return productService.findByPrice(price);

            return productService.findByPrice(price).thenApply(products
-> {
                  if (products.isEmpty()) {
                        throw new ProductNotFoundException("No products
found with price: " + price);

                  }

                  return products;

            });

      }

      @PutMapping("/product/{id}")
      public String update(@PathVariable int id, @RequestBody Product
product) {
            productService.update(product);
            return "updated Data Successfully";
      }

      @DeleteMapping("/product/{id}")
      public String delete(@PathVariable int id) {
            productService.delete(id);
            return "Successfully  Data.. is Deleted";
      }

}
```

2.6 Configuration

The AsyncConfiguration class customizes the thread pool for asynchronous tasks.

```
package com.demo.project.config;

import java.util.concurrent.Executor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

@Configuration
public class AsyncConfiguration {

    @Bean(name = "taskExecutor")
    public Executor taskExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(5);
        executor.setMaxPoolSize(10);
        executor.setQueueCapacity(25);
        executor.setThreadNamePrefix("AsyncExecutor-");
        executor.initialize();
        return executor;
    }
}
```

3. Features

CRUD Operations: Create, retrieve, update, and delete product details.

Asynchronous Processing: Non-blocking methods for enhanced performance.

Caching: Uses Redis for efficient data retrieval.

Retry Mechanism: Retry failed operations with configurable backoff.

SSL Support: Secure connections using keystore.

4. API Endpoints

Method	Endpoint	Description
POST	/api/product	Create a new product.
GET	/api/product	Retrieve all products.
GET	/api/product/{id}	Retrieve product by ID.
PUT	/api/product/{id}	Update product by ID.
DELETE	/api/product/{id}	Delete product by ID.

Command and URL's to perform CURD Operations

- **To Post Data**
  URL: https://localhost:8088/api/product

```
{
 "productName":"PUMA-Running Shoe",
 "price":999
}
```

- ➢ **To Get Data**
  URL: https://localhost:8088/api/product

```
[
    {
        "id": 1,
        "productName": "PUMA-Running Shoe",
        "price": 999
    },
    {
        "id": 2,
        "productName": "PUMA-Casual Shoe",
        "price": 799
    },
    {
        "id": 3,
        "productName": "ADIDAS-Casual Shoe",
        "price": 1300
    },
]
```

- ➢ **To Get Data by Id**
  URL: https://localhost:8088/api/product/id

```
    {
        "id": 3,
        "productName": "ADIDAS-Casual Shoe",
        "price": 1300
    }
```

- ➢ **To Get Data by ProductName**
  URL: https://localhost:8088/api/products/productName

```
    {
        "id": 2,
        "productName": "ADIDAS-Casual Shoe",
        "price": 1300
    },
```

- **To Get Data by ProductPrice**
  URL: https://localhost:8088/api/price

```
{
        "id": 1,
        "productName": "PUMA-Running Shoe",
        "price": 999
    }
```

- **To Put Data**
  URL: https://localhost:8088/api/product/id

```
        "id": 1,
        "productName": "PUMA-Running Shoe",
        "price": 999
    }
```

- **To Delete Data by Id**
  URL: https://localhost:8088/api/product/id

5. Exception Handling

The application includes a ProductNotFoundException for handling cases where no data is found.

```java
package com.demo.project.service;

public class ProductNotFoundException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public ProductNotFoundException(String message) {
        super(message);
    }

    public ProductNotFoundException(String message, Throwable cause) {
        super(message, cause);
    }
}
```