perplexity

use maven

Below is a complete Spring Reactive Maven project structure with code files matching your pricing SSE requirements. You can create the project by copying each file to appropriate locations in your Maven project folder.

Project structure

```
pricing-sse-reactive/
├── pom.xml
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/example/pricing/
│   │   │       ├── PricingSseApplication.java
│   │   │       ├── config/
│   │   │       │   └── RedisConfig.java
│   │   │       ├── controller/
│   │   │       │   └── PricingSseController.java
│   │   │       ├── model/
│   │   │       │   ├── PlaceStatus.java
│   │   │       │   └── PlaceStatusRecord.java
│   │   │       ├── publisher/
│   │   │       │   └── PricingSsePublisher.java
│   │   │       ├── service/
│   │   │       │   ├── DeadLetterConsumer.java
│   │   │       │   ├── PlaceStatusService.java
│   │   │       │   └── RedisKeyExpiryListener.java
│   │   │       └── dto/
│   │   │           └── PlaceExpiredMessage.java
│   │   └── resources/
│   │       └── application.yml
```

pom.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
           http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
```

```
    <artifactId>pricing-sse-reactive</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <name>Pricing SSE Reactive</name>
    <description>Reactive pricing status SSE notification system</description>

    <properties>
        <java.version>17</java.version>
        <spring.boot.version>3.0.5</spring.boot.version>
        <spring.kafka.version>3.0.5</spring.kafka.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>${spring.boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <!-- Spring Boot Starter WebFlux -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-webflux</artifactId>
        </dependency>

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

        <!-- Spring Kafka -->
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka</artifactId>
            <version>${spring.kafka.version}</version>
        </dependency>

        <!-- Reactor Kafka -->
        <dependency>
            <groupId>io.projectreactor.kafka</groupId>
            <artifactId>reactor-kafka</artifactId>
            <version>1.3.11</version>
        </dependency>

        <!-- Jackson for JSON -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
```

```
        </dependency>

        <!-- Lombok -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>

        <!-- Test -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <!-- Compiler plugin -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.10.1</version>
                <configuration>
                    <source>${java.version}</source>
                    <target>${java.version}</target>
                    <parameters>true</parameters>
                </configuration>
            </plugin>

            <!-- Spring Boot plugin -->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <mainClass>com.example.pricing.PricingSseApplication</mainClass>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

application.yml (src/main/resources/application.yml)

```
spring:
  redis:
    host: localhost
    port: 6379
    keyspace:
      notifications: enabled
  kafka:
    bootstrap-servers: localhost:9092
    consumer:
      group-id: pricing-expiry-group
      auto-offset-reset: earliest
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer
    producer:
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: org.springframework.kafka.support.serializer.JsonSerializer

pricing:
  expiry:
    seconds: 60
```

PricingSseApplication.java

```
package com.example.pricing;

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

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

RedisConfig.java

```
package com.example.pricing.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.repository.configuration.EnableRedisRepositories;

@Configuration
@EnableRedisRepositories(enableKeyspaceEvents = EnableRedisRepositories.EnableKeyspaceEve
public class RedisConfig {}
```

PlaceStatus.java

```
package com.example.pricing.model;

public enum PlaceStatus {
    INITIAL,
    PENDING_DRAFT,
    DRAFT;
}
```

PlaceStatusRecord.java

```
package com.example.pricing.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.Instant;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class PlaceStatusRecord {
    private String quoteId;
    private String placeId;
    private PlaceStatus status;
    private Instant lastUpdated;
}
```

PlaceExpiredMessage.java

```
package com.example.pricing.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class PlaceExpiredMessage {
    private String quoteId;
    private String placeId;
}
```

PlaceStatusService.java

```
package com.example.pricing.service;

import com.example.pricing.model.PlaceStatus;
import com.example.pricing.model.PlaceStatusRecord;
import lombok.RequiredArgsConstructor;
import org.springframework.data.redis.core.ReactiveRedisTemplate;
import org.springframework.stereotype.Service;
import reactor.core.publisher.Mono;

import java.time.Duration;
import java.time.Instant;

@Service
@RequiredArgsConstructor
public class PlaceStatusService {

    private final ReactiveRedisTemplate<String, PlaceStatusRecord> redis;

    public Mono<Void> setInitialStatus(String quoteId, String placeId, Duration ttl) {
        String key = "place:" + quoteId + ":" + placeId;
        PlaceStatusRecord record = new PlaceStatusRecord(quoteId, placeId, PlaceStatus.IN
        return redis.opsForValue().set(key, record, ttl);
    }

    public Mono<Void> updatePendingDraftStatus(String quoteId, String placeId) {
        String key = "place:" + quoteId + ":" + placeId;
        PlaceStatusRecord record = new PlaceStatusRecord(quoteId, placeId, PlaceStatus.PE
        // Set without TTL for pending draft since producer controls this
        return redis.opsForValue().set(key, record);
    }

    public Mono<Void> updateDraftStatusAndCancelExpiry(String quoteId, String placeId) {
        String key = "place:" + quoteId + ":" + placeId;
        // Clear TTL by setting key without expiration
        PlaceStatusRecord record = new PlaceStatusRecord(quoteId, placeId, PlaceStatus.DR
        return redis.opsForValue().set(key, record);
    }
}
```

RedisKeyExpiryListener.java

```
package com.example.pricing.service;

import com.example.pricing.dto.PlaceExpiredMessage;
import com.example.pricing.model.PlaceStatusRecord;
import lombok.RequiredArgsConstructor;
import org.springframework.context.ApplicationListener;
import org.springframework.data.redis.core.ReactiveRedisTemplate;
import org.springframework.data.redis.repository.events.RedisKeyExpiredEvent;
import org.springframework.kafka.core.ReactiveKafkaProducerTemplate;
```

```
import org.springframework.stereotype.Component;
import reactor.core.publisher.Mono;

@Component
@RequiredArgsConstructor
public class RedisKeyExpiryListener implements ApplicationListener<RedisKeyExpiredEvent<P

    private final ReactiveKafkaProducerTemplate<String, PlaceExpiredMessage> kafkaProduce

    @Override
    public void onApplicationEvent(RedisKeyExpiredEvent<PlaceStatusRecord> event) {
        PlaceStatusRecord expiredRecord = event.getValue();
        if (expiredRecord != null) {
            PlaceExpiredMessage msg = new PlaceExpiredMessage(expiredRecord.getQuoteId(),
            kafkaProducer.send("pricing-dead-letter-topic", msg.getQuoteId(), msg)
                    .subscribe();
        }
    }
}
```

DeadLetterConsumer.java

```
package com.example.pricing.service;

import com.example.pricing.dto.PlaceExpiredMessage;
import com.example.pricing.model.PlaceStatus;
import com.example.pricing.model.PlaceStatusRecord;
import lombok.RequiredArgsConstructor;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;
import reactor.core.publisher.Flux;

import java.util.List;

@Service
@RequiredArgsConstructor
public class DeadLetterConsumer {

    private final PlaceStatusRepository repository;
    private final PricingSsePublisher ssePublisher;

    @KafkaListener(topics = "pricing-dead-letter-topic", groupId = "pricing-expiry-group'
    public void processExpiredPlace(PlaceExpiredMessage message) {
        repository.findByQuoteId(message.getQuoteId())
                .collectList()
                .subscribe(places -> evaluateAndNotify(message.getQuoteId(), places));
    }

    private void evaluateAndNotify(String quoteId, List<PlaceStatusRecord> places) {
        boolean allDraft = places.stream().allMatch(p -> p.getStatus() == PlaceStatus.DRA
        boolean anyPending = places.stream().anyMatch(p -> p.getStatus() != PlaceStatus.D

        if (allDraft) {
            ssePublisher.sendEvent(quoteId, "pricingCompletionEvent", "Pricing completed
```

```
        } else if (anyPending) {
            ssePublisher.sendEvent(quoteId, "pendingPricingEvent", "Some places expired f
        }
    }
}
```

PricingSsePublisher.java

```
package com.example.pricing.publisher;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.mvc.method.annotation.SseEmitter;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Sinks;
import org.springframework.http.codec.ServerSentEvent;

import java.util.Map;
import java.util.Optional;
import java.util.concurrent.ConcurrentHashMap;

@Component
public class PricingSsePublisher {

    private final Map<String, Sinks.Many<ServerSentEvent<String>>> quoteSinks = new Concu

    public Flux<ServerSentEvent<String>> subscribe(String quoteId) {
        Sinks.Many<ServerSentEvent<String>> sink = Sinks.many().multicast().directBestEff
        quoteSinks.put(quoteId, sink);
        return sink.asFlux()
                .doFinally(signal -> quoteSinks.remove(quoteId));
    }

    public void sendEvent(String quoteId, String event, String data) {
        Optional.ofNullable(quoteSinks.get(quoteId))
                .ifPresent(sink -> sink.tryEmitNext(ServerSentEvent.builder().event(event
    }
}
```

PricingSseController.java

```
package com.example.pricing.controller;

import com.example.pricing.publisher.PricingSsePublisher;
import lombok.RequiredArgsConstructor;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;
import org.springframework.http.codec.ServerSentEvent;
import reactor.core.publisher.Flux;

@RestController
@RequestMapping("/pricing-sse")
```

```
@RequiredArgsConstructor
public class PricingSseController {

    private final PricingSsePublisher ssePublisher;

    @GetMapping(value = "/subscribe/{quoteId}", produces = MediaType.TEXT_EVENT_STREAM_VA
    public Flux<ServerSentEvent<String>> subscribeToQuote(@PathVariable String quoteId) {
        return ssePublisher.subscribe(quoteId);
    }
}
```

Next Steps

- Create the folder hierarchy locally.
- Copy each code snippet into respective `.java` or `.yml` files.
- Update Kafka and Redis connection settings in `application.yml` as needed.
- Build and run with Maven: `./mvnw spring-boot:run` or `mvn spring-boot:run`.

If preferred, I can prepare a GitHub Gist with all these files for easier copy-pasting. Would you like that?