package com.example.testcase;

import org.apache.camel.Exchange;

import java.util.ArrayList;

import java.util.List;

public class TestCase {

private final String id;

private final List<TestStep> steps = new ArrayList<>();

public TestCase(String id) {

this.id = id;

}

public TestCase addStep(TestStep step) {

steps.add(step);

return this;

}

public void executeSteps(Exchange exchange) {

for (TestStep step : steps) {

step.execute(exchange);

}

}

public String getId() {

return id;

}

public List<TestStep> getSteps() {

return steps;

}

}

package com.example.testcase;

import org.apache.camel.Exchange;

public interface TestStep {

void execute(Exchange exchange);

}

package com.example.testcase;

import org.apache.camel.Exchange;

import org.springframework.jdbc.core.JdbcTemplate;

public class DatabaseStep implements TestStep {

private final String tableName;

private final String operation;

public DatabaseStep(String tableName, String operation) {

this.tableName = tableName;

this.operation = operation;

}

@Override

public void execute(Exchange exchange) {

JdbcTemplate jdbcTemplate = exchange.getContext().getRegistry().lookupByNameAndType("jdbcTemplate", JdbcTemplate.class);

String sql = String.format("%s FROM %s", operation, tableName);

jdbcTemplate.execute(sql);

System.out.println("Executed DatabaseStep: " + sql);

}

}

package com.example.testcase;

import org.apache.camel.Exchange;

public class RemoteProcessStep implements TestStep {

@Override

public void execute(Exchange exchange) {

System.out.println("Executing RemoteProcessStep...");

// Simulate remote process call

try {

Thread.sleep(1000); // Simulate delay

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

System.out.println("RemoteProcessStep completed.");

}

}

package com.example.testcase;

import org.apache.camel.Exchange;

public class FileMovementStep implements TestStep {

@Override

public void execute(Exchange exchange) {

System.out.println("Executing FileMovementStep...");

// Simulate file movement

try {

Thread.sleep(500); // Simulate delay

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

System.out.println("FileMovementStep completed.");

}

}

package com.example.testcase;

import com.fasterxml.jackson.core.type.TypeReference;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;

import java.io.IOException;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

@Component

public class TestCaseRegistry {

private final Map<String, TestCase> testCases = new ConcurrentHashMap<>();

@Autowired

private JdbcTemplate jdbcTemplate;

@PostConstruct

void initializeTestCases() {

String sql = "SELECT test\_case\_id, step\_order, step\_type, step\_config FROM test\_case\_config ORDER BY test\_case\_id, step\_order";

jdbcTemplate.query(sql, rs -> {

String testCaseId = rs.getString("test\_case\_id");

String stepType = rs.getString("step\_type");

String stepConfigJson = rs.getString("step\_config");

TestCase testCase = testCases.computeIfAbsent(testCaseId, TestCase::new);

TestStep step = createStep(stepType, stepConfigJson);

testCase.addStep(step);

});

}

private TestStep createStep(String stepType, String stepConfigJson) {

Map<String, Object> stepConfig = parseStepConfig(stepConfigJson);

switch (stepType) {

case "DatabaseStep":

return new DatabaseStep(

(String) stepConfig.get("tableName"),

(String) stepConfig.get("operation")

);

case "RemoteProcessStep":

return new RemoteProcessStep();

case "FileMovementStep":

return new FileMovementStep();

default:

throw new IllegalArgumentException("Unknown step type: " + stepType);

}

}

private Map<String, Object> parseStepConfig(String stepConfigJson) {

ObjectMapper objectMapper = new ObjectMapper();

try {

return objectMapper.readValue(stepConfigJson, new TypeReference<Map<String, Object>>() {});

} catch (IOException e) {

throw new RuntimeException("Failed to parse step config JSON", e);

}

}

public TestCase getTestCase(String testCaseId) {

return testCases.get(testCaseId);

}

}

package com.example.testcase;

import org.apache.camel.builder.RouteBuilder;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

@Component

public class TestCaseRoute extends RouteBuilder {

@Autowired

private TestCaseRegistry testCaseRegistry;

@Override

public void configure() throws Exception {

// Main entry point for all test cases

from("direct:executeTestCase")

.routeId("testcase-executor")

.process(exchange -> {

String testCaseId = exchange.getIn().getHeader("testCaseId", String.class);

TestCase testCase = testCaseRegistry.getTestCase(testCaseId);

exchange.setProperty("testCase", testCase);

})

.to("direct:executeSteps");

// Execute steps sequentially in a virtual thread

from("direct:executeSteps")

.routeId("step-executor")

.process(exchange -> {

TestCase testCase = exchange.getProperty("testCase", TestCase.class);

// Execute the test case in a virtual thread

Thread.ofVirtual().start(() -> {

testCase.executeSteps(exchange);

});

});

}

}

package com.example.testcase;

import org.apache.camel.CamelContext;

import org.apache.camel.impl.DefaultCamelContext;

import org.apache.camel.impl.DefaultExchange;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

public class TestCaseApplication {

public static void main(String[] args) throws Exception {

ConfigurableApplicationContext context = SpringApplication.run(TestCaseApplication.class, args);

// Simulate test case execution

CamelContext camelContext = context.getBean(CamelContext.class);

for (int i = 0; i < 10; i++) {

String testCaseId = "testCase" + i;

Thread.ofVirtual().start(() -> {

DefaultExchange exchange = new DefaultExchange(camelContext);

exchange.getIn().setHeader("testCaseId", testCaseId);

camelContext.createProducerTemplate().send("direct:executeTestCase", exchange);

});

}

}

}

CREATE TABLE test\_case\_config (

test\_case\_id VARCHAR(50),

step\_order INT,

step\_type VARCHAR(50),

step\_config TEXT

);

INSERT INTO test\_case\_config (test\_case\_id, step\_order, step\_type, step\_config)

VALUES

('testCase1', 1, 'DatabaseStep', '{"tableName": "users", "operation": "SELECT \*"}'),

('testCase1', 2, 'RemoteProcessStep', '{}'),

('testCase2', 1, 'FileMovementStep', '{}');

Updated

package com.example.testcase;

import org.apache.camel.Exchange;

import org.springframework.jdbc.core.JdbcTemplate;

public class DatabaseStep implements TestStep {

private final String tableName;

private final String operation;

private final String whereCondition;

public DatabaseStep(String tableName, String operation, String whereCondition) {

this.tableName = tableName;

this.operation = operation;

this.whereCondition = whereCondition;

}

@Override

public void execute(Exchange exchange) {

JdbcTemplate jdbcTemplate = exchange.getContext().getRegistry().lookupByNameAndType("jdbcTemplate", JdbcTemplate.class);

String sql = buildSqlQuery();

jdbcTemplate.execute(sql);

System.out.println("Executed DatabaseStep: " + sql);

}

private String buildSqlQuery() {

StringBuilder sql = new StringBuilder();

sql.append(operation).append(" ").append(tableName);

if (whereCondition != null && !whereCondition.isEmpty()) {

sql.append(" WHERE ").append(whereCondition);

}

return sql.toString();

}

}

private TestStep createStep(String stepType, String stepConfigJson) {

Map<String, Object> stepConfig = parseStepConfig(stepConfigJson);

switch (stepType) {

case "DatabaseStep":

return new DatabaseStep(

(String) stepConfig.get("tableName"),

(String) stepConfig.get("operation"),

(String) stepConfig.get("whereCondition") // Add whereCondition

);

case "RemoteProcessStep":

return new RemoteProcessStep();

case "FileMovementStep":

return new FileMovementStep();

default:

throw new IllegalArgumentException("Unknown step type: " + stepType);

}

}

CREATE TABLE test\_case\_config (

test\_case\_id VARCHAR(50),

step\_order INT,

step\_type VARCHAR(50),

step\_config TEXT

);

{

"tableName": "users",

"operation": "SELECT \*",

"whereCondition": "age > 30"

}

After step lookup to table

CREATE TABLE test\_step\_config (

step\_type VARCHAR(50) PRIMARY KEY, -- e.g., DatabaseStep, FileMovementStep

step\_class VARCHAR(255), -- Fully qualified class name

config\_template TEXT -- JSON template for step configuration

);

INSERT INTO test\_step\_config (step\_type, step\_class, config\_template)

VALUES

('DatabaseStep', 'com.example.testcase.DatabaseStep', '{"tableName": "", "operation": "", "whereCondition": ""}'),

('FileMovementStep', 'com.example.testcase.FileMovementStep', '{"sourcePath": "", "destinationPath": ""}'),

('RemoteProcessStep', 'com.example.testcase.RemoteProcessStep', '{}');

package com.example.testcase;

import com.fasterxml.jackson.core.type.TypeReference;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

@Component

public class TestCaseRegistry {

private final Map<String, TestCase> testCases = new ConcurrentHashMap<>();

private final Map<String, TestStepConfig> stepConfigs = new ConcurrentHashMap<>();

@Autowired

private JdbcTemplate jdbcTemplate;

@PostConstruct

void initializeTestCases() {

loadStepConfigs();

loadTestCases();

}

private void loadStepConfigs() {

String sql = "SELECT step\_type, step\_class, config\_template FROM test\_step\_config";

jdbcTemplate.query(sql, rs -> {

String stepType = rs.getString("step\_type");

String stepClass = rs.getString("step\_class");

String configTemplate = rs.getString("config\_template");

stepConfigs.put(stepType, new TestStepConfig(stepType, stepClass, configTemplate));

});

}

private void loadTestCases() {

String sql = "SELECT test\_case\_id, step\_order, step\_type, step\_config FROM test\_case\_config ORDER BY test\_case\_id, step\_order";

jdbcTemplate.query(sql, rs -> {

String testCaseId = rs.getString("test\_case\_id");

String stepType = rs.getString("step\_type");

String stepConfigJson = rs.getString("step\_config");

TestCase testCase = testCases.computeIfAbsent(testCaseId, TestCase::new);

TestStep step = createStep(stepType, stepConfigJson);

testCase.addStep(step);

});

}

private TestStep createStep(String stepType, String stepConfigJson) {

TestStepConfig stepConfig = stepConfigs.get(stepType);

if (stepConfig == null) {

throw new IllegalArgumentException("Unknown step type: " + stepType);

}

Map<String, Object> config = parseStepConfig(stepConfigJson);

try {

Class<?> clazz = Class.forName(stepConfig.getStepClass());

return (TestStep) clazz.getConstructor(Map.class).newInstance(config);

} catch (Exception e) {

throw new RuntimeException("Failed to create step: " + stepType, e);

}

}

private Map<String, Object> parseStepConfig(String stepConfigJson) {

ObjectMapper objectMapper = new ObjectMapper();

try {

return objectMapper.readValue(stepConfigJson, new TypeReference<Map<String, Object>>() {});

} catch (Exception e) {

throw new RuntimeException("Failed to parse step config JSON", e);

}

}

public TestCase getTestCase(String testCaseId) {

return testCases.get(testCaseId);

}

private static class TestStepConfig {

private final String stepType;

private final String stepClass;

private final String configTemplate;

public TestStepConfig(String stepType, String stepClass, String configTemplate) {

this.stepType = stepType;

this.stepClass = stepClass;

this.configTemplate = configTemplate;

}

public String getStepType() {

return stepType;

}

public String getStepClass() {

return stepClass;

}

public String getConfigTemplate() {

return configTemplate;

}

}

}

package com.example.testcase;

import org.apache.camel.Exchange;

import org.springframework.jdbc.core.JdbcTemplate;

import java.util.Map;

public class DatabaseStep implements TestStep {

private final String tableName;

private final String operation;

private final String whereCondition;

public DatabaseStep(Map<String, Object> config) {

this.tableName = (String) config.get("tableName");

this.operation = (String) config.get("operation");

this.whereCondition = (String) config.get("whereCondition");

}

@Override

public void execute(Exchange exchange) {

JdbcTemplate jdbcTemplate = exchange.getContext().getRegistry().lookupByNameAndType("jdbcTemplate", JdbcTemplate.class);

String sql = buildSqlQuery();

jdbcTemplate.execute(sql);

System.out.println("Executed DatabaseStep: " + sql);

}

private String buildSqlQuery() {

StringBuilder sql = new StringBuilder();

sql.append(operation).append(" ").append(tableName);

if (whereCondition != null && !whereCondition.isEmpty()) {

sql.append(" WHERE ").append(whereCondition);

}

return sql.toString();

}

}

CREATE TABLE test\_case\_config (

test\_case\_id VARCHAR(50),

step\_order INT,

step\_type VARCHAR(50),

step\_config TEXT

);

INSERT INTO test\_case\_config (test\_case\_id, step\_order, step\_type, step\_config)

VALUES

('testCase1', 1, 'DatabaseStep', '{"tableName": "users", "operation": "SELECT \*", "whereCondition": "age > 30"}'),

('testCase1', 2, 'RemoteProcessStep', '{}'),

('testCase2', 1, 'FileMovementStep', '{"sourcePath": "/tmp/source.txt", "destinationPath": "/tmp/destination.txt"}');

package com.example.testcase;

import javax.persistence.\*;

@Entity

public class TestCase {

@Id

private String id;

private String description;

@Enumerated(EnumType.STRING)

private TestCaseStatus status = TestCaseStatus.PENDING; // Default status

@OneToMany(mappedBy = "testCase", cascade = CascadeType.ALL, fetch = FetchType.LAZY)

private List<TestStep> steps;

// Getters and Setters

}

enum TestCaseStatus {

PENDING,

RUNNING,

COMPLETED,

FAILED

}

package com.example.testcase;

import org.apache.camel.Exchange;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class TestCaseService {

@Autowired

private TestCaseRepository testCaseRepository;

public void executeTestCase(String testCaseId) {

TestCase testCase = testCaseRepository.findById(testCaseId).orElseThrow();

testCase.setStatus(TestCaseStatus.RUNNING);

testCaseRepository.save(testCase);

try {

// Simulate test case execution

for (TestStep step : testCase.getSteps()) {

step.execute();

}

testCase.setStatus(TestCaseStatus.COMPLETED);

} catch (Exception e) {

testCase.setStatus(TestCaseStatus.FAILED);

} finally {

testCaseRepository.save(testCase);

}

}

}