```
-- DDL Statements
CREATE TABLE TC_MASTER (
    TC_ID VARCHAR(50) PRIMARY KEY,
    TC_NAME VARCHAR(200) NOT NULL,
    DESCRIPTION TEXT,
    FLAG VARCHAR(20) NOT NULL,
    CREATED_BY VARCHAR(50),
    CREATED_DATE TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    MODIFIED_BY VARCHAR(50),
    MODIFIED_DATE TIMESTAMP
);

CREATE TABLE TC_STEPS (
    STEP_ID INT,
    TC_ID VARCHAR(50),
    STEP_NAME VARCHAR(100) NOT NULL,
    PARAMETERS TEXT,
    SEQUENCE_NO INT NOT NULL,
    STATUS VARCHAR(20),
    PRIMARY KEY (STEP_ID, TC_ID),
    FOREIGN KEY (TC_ID) REFERENCES TC_MASTER(TC_ID)
);

CREATE TABLE STEP_CONFIG (
    STEP_NAME VARCHAR(100) PRIMARY KEY,
    PARAMETER_SCHEMA TEXT NOT NULL,
    TIMEOUT_SECONDS INT NOT NULL DEFAULT 300,
    MAX_RETRIES INT NOT NULL DEFAULT 3,
    DESCRIPTION TEXT,
    CREATED_DATE TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    MODIFIED_DATE TIMESTAMP
);

CREATE TABLE TC_EXECUTION_LOG (
    EXECUTION_ID BIGINT PRIMARY KEY AUTO_INCREMENT,
    TC_ID VARCHAR(50),
    STEP_ID INT,
    START_TIME TIMESTAMP,
    END_TIME TIMESTAMP,
    STATUS VARCHAR(20),
    ERROR_MESSAGE TEXT,
    FOREIGN KEY (TC_ID) REFERENCES TC_MASTER(TC_ID)
);
```

```
-- Application Properties (application.yaml)
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/testcasedb
    username: ${DB_USERNAME}
    password: ${DB_PASSWORD}
    hikari:
      maximum-pool-size: 10
      minimum-idle: 5
      idle-timeout: 300000
  jpa:
    hibernate:
      ddl-auto: validate
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect

camel:
  springboot:
    name: TestCaseProcessor
  component:
    jdbc:
      enabled: true

logging:
  level:
    root: INFO
    com.example.testcaseprocessor: DEBUG

// Domain Models
package com.example.testcaseprocessor.model;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "TC_MASTER")
public class TestCase {
    @Id
    private String tcId;
    private String tcName;
    private String description;
    private String flag;
    private String createdBy;
    private LocalDateTime createdDate;
    private String modifiedBy;
    private LocalDateTime modifiedDate;

    // Getters, setters, and constructor
}

@Entity
@Table(name = "TC_STEPS")
@IdClass(TestCaseStepId.class)
public class TestCaseStep {
    @Id
    private Integer stepId;
    @Id
    private String tcId;
    private String stepName;
    @Column(columnDefinition = "TEXT")
    private String parameters;
    private Integer sequenceNo;
    private String status;

    // Getters, setters, and constructor
}

@Entity
@Table(name = "STEP_CONFIG")
public class StepConfiguration {
    @Id
    private String stepName;
    @Column(columnDefinition = "TEXT")
    private String parameterSchema;
    private Integer timeoutSeconds;
    private Integer maxRetries;
    private String description;
    private LocalDateTime createdDate;
    private LocalDateTime modifiedDate;

    // Getters, setters, and constructor
}

// Repositories
package com.example.testcaseprocessor.repository;

@Repository
```

```
public interface TestCaseRepository extends JpaRepository<TestCase, String> {
   List<TestCase> findByFlag(String flag);
}

@Repository
public interface TestCaseStepRepository extends JpaRepository<TestCaseStep,
TestCaseStepId> {
   List<TestCaseStep> findByTcIdOrderBySequenceNo(String tcId);
}

@Repository
public interface StepConfigRepository extends JpaRepository<StepConfiguration, String> {
}

// Services
package com.example.testcaseprocessor.service;

@Service
@Slf4j
public class TestCaseExecutionService {
   private final TestCaseStepRepository stepRepository;
   private final StepConfigRepository configRepository;
   private final TestCaseStepProcessorFactory processorFactory;
   private final ParameterValidator parameterValidator;
   private final ExecutionLogService executionLogService;

   @Autowired
   public TestCaseExecutionService(
         TestCaseStepRepository stepRepository,
         StepConfigRepository configRepository,
         TestCaseStepProcessorFactory processorFactory,
         ParameterValidator parameterValidator,
         ExecutionLogService executionLogService) {
      this.stepRepository = stepRepository;
      this.configRepository = configRepository;
      this.processorFactory = processorFactory;
      this.parameterValidator = parameterValidator;
      this.executionLogService = executionLogService;
   }

   @Transactional
   public void executeTestCase(String tcId) {
      List<TestCaseStep> steps = stepRepository.findByTcIdOrderBySequenceNo(tcId);
```

```
        for (TestCaseStep step : steps) {
            StepConfiguration config = configRepository.findById(step.getStepName())
                .orElseThrow(() -> new IllegalStateException("Step configuration not found: " +
step.getStepName()));

            executionLogService.logStepStart(tcId, step.getStepId());

            try {
                executeStep(step, config);
                executionLogService.logStepSuccess(tcId, step.getStepId());
            } catch (Exception e) {
                executionLogService.logStepFailure(tcId, step.getStepId(), e);
                throw e;
            }
        }
    }

    private void executeStep(TestCaseStep step, StepConfiguration config) {
        Map<String, String> parameters = parseParameters(step.getParameters());
        parameterValidator.validateParameters(step.getStepName(), parameters,
config.getParameterSchema());

        TestCaseStepProcessor processor = processorFactory.getProcessor(step.getStepName());
        executeWithRetry(processor, parameters, config);
    }

    private void executeWithRetry(TestCaseStepProcessor processor,
                                Map<String, String> parameters,
                                StepConfiguration config) {
        RetryConfig retryConfig = RetryConfig.<Void>custom()
            .maxAttempts(config.getMaxRetries())
            .waitDuration(Duration.ofSeconds(1))
            .retryOnException(e -> true)
            .build();

        Retry retry = RetryRegistry.of(retryConfig).retry(config.getStepName());

        retry.executeRunnable(() -> processor.processStep(parameters));
    }
}

// Camel Route
package com.example.testcaseprocessor.route;
```

```
@Component
public class TestCaseProcessorRoute extends RouteBuilder {
    private final TestCaseExecutionService executionService;
    private final VirtualThreadExecutorService virtualThreadExecutor;

    @Override
    public void configure() {
        onException(Exception.class)
            .handled(true)
            .log(LoggingLevel.ERROR, "Error processing test case: ${exception.message}")
            .process(this::handleError);

        from("sql:SELECT TC_ID FROM TC_MASTER WHERE FLAG =
'ENABLED'?delay=5000")
            .routeId("testCaseProcessor")
            .split(body())
            .process(exchange -> {
                String tcId = exchange.getIn().getBody(Map.class).get("TC_ID").toString();
                virtualThreadExecutor.executeInVirtualThread(
                    () -> executionService.executeTestCase(tcId),
                    "TestCase-" + tcId
                );
            });
    }
}

// Main Application
package com.example.testcaseprocessor;

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

    @Bean
    public ThreadFactory virtualThreadFactory() {
        return Thread.ofVirtual()
            .name("TestCase-", 0)
            .uncaughtExceptionHandler((thread, throwable) ->
                log.error("Error in thread: " + thread.getName(), throwable))
            .factory();
    }
```

```java
    @Bean
    public ExecutorService executorService(ThreadFactory virtualThreadFactory) {
        return Executors.newThreadPerTaskExecutor(virtualThreadFactory);
    }
}
```

# Test Case Processor Documentation

## Overview
The Test Case Processor is a Spring Boot application that executes test cases using virtual threads and Apache Camel. It processes test cases from a database, with each test case containing multiple steps that are executed sequentially while the test cases themselves run in parallel.

## Table of Contents

## 1. Database Schema

### TC_MASTER
Primary table for storing test cases.

```sql
CREATE TABLE TC_MASTER (
    TC_ID VARCHAR(50) PRIMARY KEY,          -- Unique identifier for the test case
    TC_NAME VARCHAR(200) NOT NULL,          -- Descriptive name of the test case
    DESCRIPTION TEXT,                       -- Detailed description
    FLAG VARCHAR(20) NOT NULL,              -- Status flag (ENABLED/DISABLED)
    CREATED_BY VARCHAR(50),                 -- User who created the test case
    CREATED_DATE TIMESTAMP,                 -- Creation timestamp
    MODIFIED_BY VARCHAR(50),                -- User who last modified the test case
    MODIFIED_DATE TIMESTAMP                 -- Last modification timestamp
);
```

### TC_STEPS
Stores individual steps for each test case.

```sql
CREATE TABLE TC_STEPS (
   STEP_ID INT,                          -- Step identifier
   TC_ID VARCHAR(50),                    -- Reference to TC_MASTER
   STEP_NAME VARCHAR(100) NOT NULL,      -- Name of the step
   PARAMETERS TEXT,                      -- JSON parameters for the step
   SEQUENCE_NO INT NOT NULL,             -- Execution order
   STATUS VARCHAR(20),                   -- Current status of the step
   PRIMARY KEY (STEP_ID, TC_ID)
);
```

### STEP_CONFIG
Configuration table for step definitions.

```sql
CREATE TABLE STEP_CONFIG (
   STEP_NAME VARCHAR(100) PRIMARY KEY,    -- Step type identifier
   PARAMETER_SCHEMA TEXT NOT NULL,        -- JSON schema for parameter validation
   TIMEOUT_SECONDS INT NOT NULL,          -- Maximum execution time
   MAX_RETRIES INT NOT NULL,              -- Maximum retry attempts
   DESCRIPTION TEXT,                      -- Step description
   CREATED_DATE TIMESTAMP,                -- Creation timestamp
   MODIFIED_DATE TIMESTAMP                -- Last modification timestamp
);
```

### TC_EXECUTION_LOG
Logs execution details for auditing and monitoring.

```sql
CREATE TABLE TC_EXECUTION_LOG (
   EXECUTION_ID BIGINT PRIMARY KEY,       -- Unique execution identifier
   TC_ID VARCHAR(50),                     -- Reference to TC_MASTER
   STEP_ID INT,                           -- Reference to TC_STEPS
   START_TIME TIMESTAMP,                  -- Step start time
   END_TIME TIMESTAMP,                    -- Step end time
   STATUS VARCHAR(20),                    -- Execution status
   ERROR_MESSAGE TEXT                     -- Error details if failed
```

);

## 2. Core Components

### Domain Models

#### TestCase.java
```java
@Entity
@Table(name = "TC_MASTER")
public class TestCase {
    // Represents a test case entity
    // Contains test case metadata and status
}
```

#### TestCaseStep.java
```java
@Entity
@Table(name = "TC_STEPS")
public class TestCaseStep {
    // Represents an individual step within a test case
    // Contains step parameters and execution order
}
```

### Services

#### TestCaseExecutionService
Primary service for test case execution.

Key responsibilities:
- Orchestrates test case execution
- Manages step sequencing
- Handles parameter validation
- Implements retry logic

```java
@Service
public class TestCaseExecutionService {
    /**
     * Executes a test case with the given ID
     * @param tcId Test case identifier
     * @throws IllegalStateException if step configuration is missing
     */
    public void executeTestCase(String tcId) {
        // Implementation details
    }
}
```

#### ParameterValidator
Validates step parameters against JSON schemas.

```java
@Component
public class ParameterValidator {
    /**
     * Validates parameters against schema
     * @param stepName Name of the step
     * @param parameters Parameter map
     * @param schema JSON schema
     * @throws IllegalArgumentException if validation fails
     */
    public void validateParameters(String stepName,
                                  Map<String, String> parameters,
                                  String schema) {
        // Implementation details
    }
}
```

### Camel Routes

#### TestCaseProcessorRoute
Main route for processing test cases.

```java
@Component
public class TestCaseProcessorRoute extends RouteBuilder {
    @Override
    public void configure() {
        // Polls database for enabled test cases
        // Processes each test case in a virtual thread
        // Handles errors and logging
    }
}
```

## 3. Configuration

### application.yaml
```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/testcasedb
    username: ${DB_USERNAME}
    password: ${DB_PASSWORD}
    hikari:
      maximum-pool-size: 10
      minimum-idle: 5
```

### Virtual Thread Configuration
```java
@Configuration
public class TestCaseProcessorConfig {
    @Bean
    public ThreadFactory virtualThreadFactory() {
        // Configures virtual thread factory for test case execution
    }
}
```

## 4. Application Flow

1. **Test Case Discovery**
   - Camel route polls TC_MASTER table for enabled test cases
   - Each test case is processed independently

2. **Step Execution**
   - Steps are retrieved and ordered by sequence number
   - Parameters are validated against JSON schema
   - Step is executed with retry logic
   - Results are logged

3. **Monitoring**
   - Execution progress is tracked in TC_EXECUTION_LOG
   - Errors are captured and logged
   - Metrics are collected for monitoring

## 5. Error Handling

### Retry Mechanism
```java
private void executeWithRetry(TestCaseStepProcessor processor,
                              Map<String, String> parameters,
                              StepConfiguration config) {
    // Implements exponential backoff
    // Respects maximum retry attempts
    // Honors timeout configuration
}
```

### Error Logging
- All errors are captured in TC_EXECUTION_LOG
- Stack traces are preserved for debugging
- Error notifications can be configured

## 6. Monitoring

### Metrics Collection
- Step execution times
- Success/failure rates
- Retry attempts
- Resource utilization

### Logging
```java
logging:
  level:
    root: INFO
    com.example.testcaseprocessor: DEBUG
```

## 7. Installation & Setup

1. **Database Setup**
   ```bash
   psql -U postgres -d testcasedb -f schema.sql
   ```

2. **Application Configuration**
   - Configure database connection
   - Set logging levels

   - Configure thread pools

3. **Deployment**
   ```bash
   ./mvnw clean package
   java -jar target/test-case-processor.jar
   ```

## 8. Usage Examples

### Adding a Test Case
```sql
INSERT INTO TC_MASTER (TC_ID, TC_NAME, FLAG)
VALUES ('TC001', 'Database Cleanup', 'ENABLED');

INSERT INTO TC_STEPS (STEP_ID, TC_ID, STEP_NAME, PARAMETERS, SEQUENCE_NO)
VALUES (1, 'TC001', 'DELETE_INSERT_AIT_SCAN_WINDOW',
        '{"aitNumber": "AIT123456"}', 1);
```

### Step Configuration
```sql
INSERT INTO STEP_CONFIG (STEP_NAME, PARAMETER_SCHEMA,
TIMEOUT_SECONDS, MAX_RETRIES)
VALUES ('DELETE_INSERT_AIT_SCAN_WINDOW',
        '{"type":"object","required":["aitNumber"]}',
        300, 3);
```

## Best Practices

1. **Parameter Validation**
   - Always provide JSON schemas for parameters
   - Include pattern validation where appropriate
   - Document parameter constraints

2. **Error Handling**
   - Configure appropriate retry counts
   - Set realistic timeouts
   - Monitor error patterns

3. **Performance**
   - Use appropriate thread pool sizes
   - Monitor database connection pool

   - Index frequently queried columns

## Maintenance

1. **Database Maintenance**
   - Regular cleanup of execution logs
   - Index maintenance
   - Statistics collection

2. **Monitoring**
   - Watch for failed executions
   - Monitor resource utilization
   - Track execution times

3. **Troubleshooting**
   - Check execution logs
   - Verify parameter validation
   - Review retry patterns