Code Updates for callicoder/spring-boot-flyway-example with Rollback Framework

Overview

The existing repository is a basic Spring Boot + Flyway example. Here are the comprehensive updates needed to integrate the rollback framework with H2 support for local testing.

1. Update pom.xml

Replace the existing pom.xml with:

xml

```xml
<?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
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>flyway-demo</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>flyway-demo</name>
    <description>Spring Boot Flyway Demo with Rollback Support</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.0</version>
        <relativePath/>
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>17</java.version>
        <flyway.version>10.0.0</flyway.version>
    </properties>

    <dependencies>
        <!-- Spring Boot Starters -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
```

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

<!-- Flyway -->
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
    <version>${flyway.version}</version>
</dependency>
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-mysql</artifactId>
    <version>${flyway.version}</version>
</dependency>

<!-- Databases -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>

<!-- Utilities -->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
</dependency>

<!-- Monitoring -->
```

```
    <dependency>
      <groupId>io.micrometer</groupId>
      <artifactId>micrometer-registry-prometheus</artifactId>
    </dependency>

    <!-- Testing -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.testcontainers</groupId>
      <artifactId>testcontainers</artifactId>
      <version>1.19.0</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.testcontainers</groupId>
      <artifactId>mysql</artifactId>
      <version>1.19.0</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <excludes>
            <exclude>
              <groupId>org.projectlombok</groupId>
              <artifactId>lombok</artifactId>
            </exclude>
          </excludes>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.flywaydb</groupId>
        <artifactId>flyway-maven-plugin</artifactId>
        <version>${flyway.version}</version>
      </plugin>
    </plugins>
```

```
    </build>
</project>
```

2. Update Application Properties

Replace `src/main/resources/application.properties` with `application.yml`:

yaml

```yaml
# src/main/resources/application.yml
spring:
  application:
    name: flyway-demo

  profiles:
    active: ${SPRING_PROFILES_ACTIVE:local}

  jpa:
    show-sql: false
    hibernate:
      ddl-auto: validate
    properties:
      hibernate:
        format_sql: true

  flyway:
    enabled: true
    baseline-on-migrate: true
    locations:
      - classpath:db/migration
      - classpath:db/rollback
    validate-on-migrate: true

# Rollback Configuration
flyway:
  rollback:
    enabled: true
    auto-rollback-on-failure: false
    require-approval: false
    snapshot:
      enabled: true
      storage-path: ${user.home}/flyway-snapshots
      retention-days: 7
    audit:
      enabled: true
      table-name: flyway_rollback_audit

# Actuator
management:
  endpoints:
    web:
      exposure:
        include: health,info,metrics,flyway

# Logging
```

```yaml
logging:
  level:
    com.example: DEBUG
    org.flywaydb: DEBUG

---
# Local Profile (H2)
spring:
  config:
    activate:
      on-profile: local

  datasource:
    url: jdbc:h2:mem:flyway_demo;MODE=MySQL;DATABASE_TO_LOWER=TRUE;DEFAULT_NULL_ORDERING=HIGH
    username: sa
    password:
    driver-class-name: org.h2.Driver

  h2:
    console:
      enabled: true
      path: /h2-console

  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.H2Dialect

---
# MySQL Profile
spring:
  config:
    activate:
      on-profile: mysql

  datasource:
    url: jdbc:mysql://localhost:3306/flyway_demo?useSSL=false&serverTimezone=UTC
    username: root
    password: ${MYSQL_PASSWORD:callicoder}
    driver-class-name: com.mysql.cj.jdbc.Driver

  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL8Dialect

---
```

```yaml
# Test Profile
spring:
  config:
    activate:
      on-profile: test

  datasource:
    url: jdbc:h2:mem:testdb;MODE=MySQL
    driver-class-name: org.h2.Driver

  flyway:
    rollback:
      snapshot:
        storage-path: ${java.io.tmpdir}/test-snapshots
```

3. Update Main Application Class

java

```java
// src/main/java/com/example/FlywayDemoApplication.java
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.properties.ConfigurationPropertiesScan;
import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling
@ConfigurationPropertiesScan
public class FlywayDemoApplication {

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

4. Add Rollback Framework Components

4.1 Configuration Classes

java

```java
// src/main/java/com/example/config/FlywayRollbackConfiguration.java
package com.example.config;

import com.example.rollback.FlywayRollbackManager;
import com.example.rollback.properties.FlywayRollbackProperties;
import lombok.extern.slf4j.Slf4j;
import org.flywaydb.core.Flyway;
import org.springframework.boot.autoconfigure.flyway.FlywayMigrationStrategy;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;

import javax.sql.DataSource;

@Configuration
@EnableConfigurationProperties(FlywayRollbackProperties.class)
@Slf4j
public class FlywayRollbackConfiguration {

    @Bean
    public FlywayRollbackManager flywayRollbackManager(
            DataSource dataSource,
            FlywayRollbackProperties properties) {
        return new FlywayRollbackManager(dataSource, properties);
    }

    @Bean
    public FlywayMigrationStrategy flywayMigrationStrategy(FlywayRollbackManager rollbackManager) {
        return flyway -> {
            log.info("Starting Flyway migration with rollback support");

            // Create pre-migration snapshot
            if (rollbackManager.isSnapshotEnabled()) {
                try {
                    String snapshotId = rollbackManager.createPreMigrationSnapshot();
                    log.info("Created pre-migration snapshot: {}", snapshotId);
                } catch (Exception e) {
                    log.warn("Failed to create pre-migration snapshot", e);
                }
            }

            // Execute migration
            try {
                flyway.migrate();
                log.info("Flyway migration completed successfully");
```

```
            } catch (Exception e) {
                log.error("Flyway migration failed", e);

                if (rollbackManager.isAutoRollbackEnabled()) {
                    log.info("Attempting automatic rollback");
                    rollbackManager.handleMigrationFailure(e);
                }

                throw e;
            }
        };
    }
}
```

java

```
// src/main/java/com/example/config/SecurityConfig.java
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            .authorizeHttpRequests(authz -> authz
                .requestMatchers("/h2-console/**").permitAll()
                .requestMatchers("/actuator/**").permitAll()
                .requestMatchers("/api/flyway/rollback/**").authenticated()
                .anyRequest().permitAll()
            )
            .headers().frameOptions().disable();

        return http.build();
    }
}
```

4.2 Rollback Properties

java

```java
// src/main/java/com/example/rollback/properties/FlywayRollbackProperties.java
package com.example.rollback.properties;

import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties(prefix = "flyway.rollback")
@Data
public class FlywayRollbackProperties {

    private boolean enabled = true;
    private boolean autoRollbackOnFailure = false;
    private boolean requireApproval = true;

    private SnapshotProperties snapshot = new SnapshotProperties();
    private AuditProperties audit = new AuditProperties();

    @Data
    public static class SnapshotProperties {
        private boolean enabled = true;
        private String storagePath = System.getProperty("user.home") + "/flyway-snapshots";
        private int retentionDays = 7;
    }

    @Data
    public static class AuditProperties {
        private boolean enabled = true;
        private String tableName = "flyway_rollback_audit";
    }
}
```

4.3 Simplified Rollback Manager

java

```java
// src/main/java/com/example/rollback/FlywayRollbackManager.java
package com.example.rollback;

import com.example.rollback.properties.FlywayRollbackProperties;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import javax.sql.DataSource;
import java.io.File;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

@Component
@Slf4j
@RequiredArgsConstructor
public class FlywayRollbackManager {

    private final DataSource dataSource;
    private final FlywayRollbackProperties properties;
    private final JdbcTemplate jdbcTemplate;

    public FlywayRollbackManager(DataSource dataSource, FlywayRollbackProperties properties) {
        this.dataSource = dataSource;
        this.properties = properties;
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public boolean isSnapshotEnabled() {
        return properties.getSnapshot().isEnabled();
    }

    public boolean isAutoRollbackEnabled() {
        return properties.isAutoRollbackOnFailure();
    }

    @Transactional
    public RollbackResult rollbackToVersion(String targetVersion) {
        String rollbackId = UUID.randomUUID().toString();
        log.info("Starting rollback {} to version {}", rollbackId, targetVersion);
```

```
    try {
        // Get current version
        String currentVersion = getCurrentVersion();
        log.info("Current version: {}", currentVersion);

        // Create snapshot before rollback
        String snapshotId = null;
        if (properties.getSnapshot().isEnabled()) {
            snapshotId = createSnapshot("rollback_" + targetVersion);
        }

        // Get rollback scripts to execute
        List<String> rollbackScripts = getRollbackScripts(currentVersion, targetVersion);

        // Execute rollback scripts
        for (String script : rollbackScripts) {
            log.info("Executing rollback script: {}", script);
            executeRollbackScript(script);
        }

        // Update flyway schema history
        updateFlywaySchemaHistory(targetVersion);

        // Audit the rollback
        auditRollback(rollbackId, targetVersion, "SUCCESS", null);

        return new RollbackResult(true, rollbackId, targetVersion, snapshotId, null);

    } catch (Exception e) {
        log.error("Rollback failed", e);
        auditRollback(rollbackId, targetVersion, "FAILED", e.getMessage());
        return new RollbackResult(false, rollbackId, targetVersion, null, e.getMessage());
    }
}

public String createPreMigrationSnapshot() {
    return createSnapshot("pre_migration");
}

public String createSnapshot(String prefix) {
    String snapshotId = prefix + "_" + LocalDateTime.now()
        .format(DateTimeFormatter.ofPattern("yyyyMMdd_HHmmss"));

    log.info("Creating snapshot: {}", snapshotId);

    try {
```

```
            // Create snapshot directory
            Path snapshotDir = Paths.get(properties.getSnapshot().getStoragePath(), snapshotId);
            Files.createDirectories(snapshotDir);

            // Get all tables
            List<String> tables = getAllTables();

            // Create snapshot for each table
            for (String table : tables) {
                createTableSnapshot(table, snapshotDir);
            }

            log.info("Snapshot {} created successfully", snapshotId);
            return snapshotId;

        } catch (Exception e) {
            log.error("Failed to create snapshot", e);
            throw new RuntimeException("Snapshot creation failed", e);
        }
    }

    private void createTableSnapshot(String tableName, Path snapshotDir) {
        try {
            // For H2, create a CSV export
            String sql = String.format("SCRIPT TO '%s/%s.sql' TABLE %s",
                snapshotDir.toString(), tableName, tableName);

            // For MySQL, you would use different approach
            if (isMySQL()) {
                // Create table copy
                jdbcTemplate.execute(String.format(
                    "CREATE TABLE snapshot_%s AS SELECT * FROM %s",
                    tableName, tableName));
            } else {
                // H2 approach - export to file
                jdbcTemplate.execute(sql);
            }

        } catch (Exception e) {
            log.warn("Failed to snapshot table: {}", tableName, e);
        }
    }

    private String getCurrentVersion() {
        try {
            return jdbcTemplate.queryForObject(
                "SELECT version FROM flyway_schema_history " +
```

```
            "WHERE success = true " +
            "ORDER BY installed_rank DESC LIMIT 1",
            String.class
        );
    } catch (Exception e) {
        return "0";
    }
}

private List<String> getRollbackScripts(String currentVersion, String targetVersion) {
    List<String> scripts = new ArrayList<>();

    // Get versions to rollback
    List<String> versionsToRollback = jdbcTemplate.queryForList(
        "SELECT version FROM flyway_schema_history " +
        "WHERE version > ? AND version <= ? " +
        "ORDER BY installed_rank DESC",
        String.class, targetVersion, currentVersion
    );

    // Find corresponding rollback scripts
    for (String version : versionsToRollback) {
        String rollbackScript = "classpath:db/rollback/U" + version + "__rollback.sql";
        scripts.add(rollbackScript);
    }

    return scripts;
}

private void executeRollbackScript(String scriptPath) {
    // In a real implementation, you would read and execute the SQL file
    // For now, we'll just log it
    log.info("Would execute rollback script: {}", scriptPath);
}

private void updateFlywaySchemaHistory(String targetVersion) {
    // Remove entries after target version
    jdbcTemplate.update(
        "DELETE FROM flyway_schema_history WHERE version > ?",
        targetVersion
    );
}

private void auditRollback(String rollbackId, String version, String status, String error) {
    if (!properties.getAudit().isEnabled()) {
        return;
    }
```

```
        try {
            jdbcTemplate.update(
                "INSERT INTO " + properties.getAudit().getTableName() +
                " (rollback_id, version, status, error_message, performed_at) " +
                "VALUES (?, ?, ?, ?, ?)",
                rollbackId, version, status, error, LocalDateTime.now()
            );
        } catch (Exception e) {
            log.warn("Failed to audit rollback", e);
        }
    }

    private List<String> getAllTables() {
        return jdbcTemplate.queryForList(
            "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES " +
            "WHERE TABLE_SCHEMA = SCHEMA() AND TABLE_TYPE = 'TABLE'",
            String.class
        );
    }

    private boolean isMySQL() {
        try {
            String url = dataSource.getConnection().getMetaData().getURL();
            return url.contains("mysql");
        } catch (Exception e) {
            return false;
        }
    }

    public void handleMigrationFailure(Exception e) {
        log.error("Handling migration failure", e);
        // Implement auto-rollback logic
    }
}
```

4.4 Rollback Controller

java

```java
// src/main/java/com/example/controller/RollbackController.java
package com.example.controller;

import com.example.rollback.FlywayRollbackManager;
import com.example.rollback.RollbackRequest;
import com.example.rollback.RollbackResult;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/flyway/rollback")
@RequiredArgsConstructor
@Slf4j
public class RollbackController {

    private final FlywayRollbackManager rollbackManager;

    @PostMapping("/execute")
    public ResponseEntity<RollbackResult> executeRollback(@RequestBody RollbackRequest request) {
        log.info("Rollback request received: {}", request);

        RollbackResult result = rollbackManager.rollbackToVersion(request.getTargetVersion());

        if (result.isSuccess()) {
            return ResponseEntity.ok(result);
        } else {
            return ResponseEntity.internalServerError().body(result);
        }
    }

    @PostMapping("/snapshot")
    public ResponseEntity<String> createSnapshot() {
        String snapshotId = rollbackManager.createSnapshot("manual");
        return ResponseEntity.ok(snapshotId);
    }
}
```

4.5 Model Classes

java

```java
// src/main/java/com/example/rollback/RollbackRequest.java
package com.example.rollback;

import lombok.Data;

@Data
public class RollbackRequest {
    private String targetVersion;
    private boolean dryRun;
    private String reason;
}

// src/main/java/com/example/rollback/RollbackResult.java
package com.example.rollback;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class RollbackResult {
    private boolean success;
    private String rollbackId;
    private String targetVersion;
    private String snapshotId;
    private String errorMessage;
}
```

5. Update Migration Scripts

5.1 Update existing migrations to be H2 compatible

sql

```sql
-- src/main/resources/db/migration/V1__init.sql
CREATE TABLE IF NOT EXISTS users (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(100) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    created_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create rollback audit table
CREATE TABLE IF NOT EXISTS flyway_rollback_audit (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    rollback_id VARCHAR(50) NOT NULL,
    version VARCHAR(50) NOT NULL,
    status VARCHAR(20) NOT NULL,
    error_message TEXT,
    performed_at TIMESTAMP NOT NULL
);
```

5.2 Add rollback scripts

sql

```sql
-- src/main/resources/db/rollback/U1__rollback_init.sql
-- Rollback script for V1__init.sql

-- Archive data before dropping
CREATE TABLE IF NOT EXISTS archive_users AS SELECT * FROM users;

-- Drop tables
DROP TABLE IF EXISTS users;
DROP TABLE IF EXISTS flyway_rollback_audit;
```

5.3 Add more example migrations

sql

```sql
-- src/main/resources/db/migration/V2__add_user_profile.sql
CREATE TABLE IF NOT EXISTS user_profiles (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    bio TEXT,
    avatar_url VARCHAR(500),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id)
);

-- Add sample data
INSERT INTO users (username, email, first_name, last_name) VALUES
('john_doe', 'john@example.com', 'John', 'Doe'),
('jane_smith', 'jane@example.com', 'Jane', 'Smith');

-- src/main/resources/db/rollback/U2__rollback_user_profile.sql
-- Archive profile data
CREATE TABLE IF NOT EXISTS archive_user_profiles AS SELECT * FROM user_profiles;

-- Drop profile table
DROP TABLE IF EXISTS user_profiles;

-- Remove sample users
DELETE FROM users WHERE username IN ('john_doe', 'jane_smith');
```

6. Add Domain Models

java

```java
// src/main/java/com/example/model/User.java
package com.example.model;

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

@Entity
@Table(name = "users")
@Data
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    private String email;

    private String firstName;
    private String lastName;

    @Column(name = "created_time")
    private LocalDateTime createdTime;

    @PrePersist
    protected void onCreate() {
        createdTime = LocalDateTime.now();
    }
}
```

7. Add Simple Test Controller

java

```java
// src/main/java/com/example/controller/UserController.java
package com.example.controller;

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

import java.util.List;

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

    private final UserRepository userRepository;

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

    @PostMapping
    public User createUser(@RequestBody User user) {
        return userRepository.save(user);
    }
}

// src/main/java/com/example/repository/UserRepository.java
package com.example.repository;

import com.example.model.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
}
```

8. Add Test Configuration

java

```java
// src/test/java/com/example/FlywayRollbackTest.java
package com.example;

import com.example.rollback.FlywayRollbackManager;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.ActiveProfiles;

import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest
@ActiveProfiles("test")
class FlywayRollbackTest {

    @Autowired
    private FlywayRollbackManager rollbackManager;

    @Test
    void testCreateSnapshot() {
        String snapshotId = rollbackManager.createSnapshot("test");
        assertThat(snapshotId).isNotNull();
    }
}
```

9. Add Docker Compose for MySQL Testing

yaml

```yaml
# docker-compose.yml
version: '3.8'

services:
  mysql:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: flyway_demo
      MYSQL_USER: demo_user
      MYSQL_PASSWORD: demo_password
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql

volumes:
  mysql_data:
```

10. Updated README.md

markdown

```
# Spring Boot Flyway Example with Rollback Support

This project demonstrates Spring Boot integration with Flyway including a comprehensive rollback framework.

## Features

- Spring Boot 3.2.0 with Java 17
- Flyway 10.0.0 for database migrations
- H2 database for local development
- MySQL support for production
- Comprehensive rollback framework
- REST API for rollback operations
- Snapshot management
- Audit logging

## Requirements

- Java 17
- Maven 3.x
- MySQL 8.x (optional, for MySQL profile)

## Quick Start

### 1. Clone the repository
```bash
git clone https://github.com/callicoder/spring-boot-flyway-example.git
cd spring-boot-flyway-example
```

2. Run with H2 (default)

bash

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

Access H2 Console: http://localhost:8080/h2-console

- JDBC URL: `jdbc:h2:mem:flyway_demo`
- Username: `sa`
- Password: (leave empty)

3. Run with MySQL
```

bash

```
# Start MySQL with Docker
docker-compose up -d

# Run application with MySQL profile
mvn spring-boot:run -Dspring.profiles.active=mysql
```

Testing Rollback

1. Check current version

bash

```
curl http://localhost:8080/actuator/flyway
```

2. Create a snapshot

bash

```
curl -X POST http://localhost:8080/api/flyway/rollback/snapshot
```

3. Execute rollback

bash

```
curl -X POST http://localhost:8080/api/flyway/rollback/execute \
  -H "Content-Type: application/json" \
  -d '{"targetVersion": "1", "reason": "Testing rollback"}'
```

API Endpoints

- `GET /api/users` - List all users
- `POST /api/users` - Create a user
- `POST /api/flyway/rollback/execute` - Execute rollback
- `POST /api/flyway/rollback/snapshot` - Create snapshot
- `GET /actuator/flyway` - Flyway migration info
- `GET /h2-console` - H2 Database console (local profile only)

Project Structure

```
src/
├── main/
│   ├── java/
│   │   └── com/example/
│   │       ├── config/       # Configuration classes
│   │       ├── controller/   # REST controllers
│   │       ├── model/        # JPA entities
│   │       ├── repository/   # Data repositories
│   │       └── rollback/     # Rollback framework
│   └── resources/
│       ├── db/
│       │   ├── migration/    # Forward migration scripts
│       │   └── rollback/     # Rollback scripts
│       └── application.yml   # Application configuration
└── test/
```

Configuration

The application supports multiple profiles:

- `local` (default): Uses H2 in-memory database
- `mysql`: Uses MySQL database
- `test`: Uses H2 for testing

License

MIT

```
## Usage Instructions

1. **Clone the repository**:
   ```bash
   git clone https://github.com/callicoder/spring-boot-flyway-example.git
   cd spring-boot-flyway-example
```

2. **Apply these updates**:
   - Replace the pom.xml
   - Delete application.properties and create application.yml
   - Add all the Java classes in their respective packages
   - Create the rollback directory structure
   - Add migration and rollback scripts
```

3. **Run with H2 (default)**:

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

4. **Test the rollback**:

```bash
# Check migrations
curl http://localhost:8080/actuator/flyway

# Create snapshot
curl -X POST http://localhost:8080/api/flyway/rollback/snapshot

# Test users API
curl http://localhost:8080/api/users
```

The updated application now includes:

- Full rollback framework integration
- H2 database for local testing
- MySQL support for production
- REST APIs for rollback operations
- Proper migration and rollback scripts
- Spring profiles for different environments
- Comprehensive testing support