

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА  
(САМАРСКИЙ УНИВЕРСИТЕТ)»

Институт Информатики и кибернетики   
Кафедра Программных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
  
к лабораторной работе №2 по дисциплине «Автоматизация тестирования»

Обучающийся группы 6231-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гижевская В.Д.

Обучающийся группы 6231-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алкеев М.Г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лобанков А.А.

Самара 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146727791)

[2 Результаты работы 4](#_Toc146727792)

[2.1 Результаты задания 1 4](#_Toc146727793)

[2.2 Результаты задания 2 4](#_Toc146727794)

[2.3 Результаты задания 3 4](#_Toc146727795)

[ПРИЛОЖЕНИЕ А Листинг программы 5](#_Toc146727796)

1. Постановка задачи

В процессе выполнения заданий ознакомиться с фреймворком для управления миграциями баз данных Flyway и фреймворком для интеграционного тестирования Testcontainers.

Задание 1

На основе первой лабораторной работы создать базу данных для хранения вычислений. Таблица должна содержать – первое число и его систему счисления, второе число и его систему счисления (числа необходимо хранить в строковом виде), дату и время вычисления, тип операции (сложение/вычитание/умножение/деление).

Задание 2

На основе первой лабораторной работы создать Spring Boot приложение, в котором реализовать следующие рест енд-поинты: 1) Получение результата сложения/вычитания/умножения/деления двух чисел. Числа могут быть в разных системах счисления. Все входные данные сохраняются в БД, в том числе время и дата выполнения запроса. 2) Получение данных о всех вычислениях за определенный момент времени, с учетом операции и используемой системы счисления для каждого из чисел. 3) Все DDL операции должны быть оформлены в виде миграций (Flyway).

Задание 3

Провести интеграционное тестирование приложения с использованием фреймворка Testcontainers. Миграции с тестовыми данными (20-30 записей) должны накатываться при запуске тестов (для каждого теста отдельно или перед запуском всех тестов)

1. Результаты работы
   1. Результаты задания 1

Была создана база данных с таблицей для хранения вычислений. Таблица содержит информацию о первом и втором числе в строковом виде, их системах счисления, дате и времени вычисления, а также типе операции (сложение, вычитание, умножение, деление).

* 1. Результаты задания 2

В классе реализованы тесты для рест енд-поинтов, проверяющие корректность выполнения арифметических операций в различных системах счисления. Ниже представлено краткое описание каждого теста.

Тест testCalculateDecimal:

* Проверяет корректность выполнения арифметических операций (сложение, умножение, деление, вычитание) в десятичной системе счисления.
* Ожидается успешный ответ с корректным результатом для каждой операции.

Тест testCalculateBinary:

* Проверяет корректность выполнения арифметических операций (сложение, умножение, деление, вычитание) в восьмеричной системе счисления.
* Ожидается успешный ответ с корректным результатом для каждой операции.

Тест testFindAllBetweenDates

* Проверяет корректность выполнения запроса для получения всех операций в определенный период времени.
* Метод использует Testcontainers для настройки временной базы данных и выполнения миграций перед проведением теста.
* Ожидается, что результат запроса совпадет с ожидаемым JSON, считанным из файла.
  1. Результаты задания 3

Интеграционное тестирование было успешно проведено с использованием фреймворка Testcontainers. Миграции с тестовыми данными (20-30 записей) накатываются перед запуском каждого теста, обеспечивая корректность и стабильность тестового окружения.

ПРИЛОЖЕНИЕ А  
Листинг программы

**AppTests:**

package com.example;

import com.example.Calculator.CalculatorRepository;

import com.example.Calculator.OperationEntity;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

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

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.test.web.servlet.MockMvc;

import org.testcontainers.shaded.com.fasterxml.jackson.databind.ObjectMapper;

import org.testcontainers.shaded.com.fasterxml.jackson.databind.ObjectWriter;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.sql.Date;

import java.util.List;

import static junit.framework.TestCase.assertEquals;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

import org.flywaydb.core.Flyway;

@SpringBootTest

@ComponentScan(basePackages = "com.example")

@AutoConfigureMockMvc

class AppTests extends AuditVizualizationBaseTest{

@Autowired

CalculatorRepository calculatorRepository;

@Autowired

private MockMvc mockMvc;

@Test

void testCalculateDecimal() throws Exception {

mockMvc.perform(get("/calculate/10/decimal/20/decimal/add"))

.andExpect(status().isOk())

.andExpect(content().string("Result: 30"))

.andDo(print());

mockMvc.perform(get("/calculate/10/decimal/20/decimal/multiply"))

.andExpect(status().isOk())

.andExpect(content().string("Result: 200"))

.andDo(print());

mockMvc.perform(get("/calculate/20/decimal/10/decimal/divide"))

.andExpect(status().isOk())

.andExpect(content().string("Result: 2"))

.andDo(print());

mockMvc.perform(get("/calculate/20/decimal/10/decimal/subtract"))

.andExpect(status().isOk())

.andExpect(content().string("Result: 10"))

.andDo(print());

}

@Test

void testCalculateBinary() throws Exception {

mockMvc.perform(get("/calculate/47/octal/13/octal/add"))

.andExpect(status().isOk())

.andExpect(content().string("Result: 62"))

.andDo(print());

mockMvc.perform(get("/calculate/56/octal/12/octal/multiply"))

.andExpect(status().isOk())

.andExpect(content().string("Result: 714"))

.andDo(print());

mockMvc.perform(get("/calculate/314/octal/23/octal/divide"))

.andExpect(status().isOk())

.andExpect(content().string("Result: 12"))

.andDo(print());

mockMvc.perform(get("/calculate/75/octal/24/octal/subtract"))

.andExpect(status().isOk())

.andExpect(content().string("Result: 51"))

.andDo(print());

}

@Test

void testFindAllBetweenDates() throws Exception {

Flyway flyway = Flyway.configure()

.dataSource(POSTGRE\_SQL\_CONTAINER.getJdbcUrl(), POSTGRE\_SQL\_CONTAINER.getUsername(), POSTGRE\_SQL\_CONTAINER.getPassword())

.locations("/db.testMigration")

.load();

flyway.repair();

flyway.migrate();

List<OperationEntity> operationEntities = calculatorRepository.findAllByDateBetweenAndOperationTypeAndFirstNumSystemAndSecondNumSystem(

Date.valueOf("2023-10-01"), Date.valueOf("2023-11-01"), "add", "decimal", "decimal");

ObjectWriter ow = new ObjectMapper().writer().withDefaultPrettyPrinter();

String json = ow.writeValueAsString(operationEntities);

String expectedJson = Files.readString(Paths.get("src/test/resources/expected.json"));

assertEquals(expectedJson, json);

}

}

**AuditVizualizationBaseTest:**

package com.example;

import com.example.Calculator.OperationEntity;

import org.flywaydb.core.Flyway;

import org.flywaydb.test.FlywayTestExecutionListener;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.util.TestPropertyValues;

import org.springframework.boot.test.web.client.TestRestTemplate;

import org.springframework.context.ApplicationContextInitializer;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.test.context.\*;

import org.springframework.test.context.jdbc.SqlScriptsTestExecutionListener;

import org.springframework.test.context.support.DependencyInjectionTestExecutionListener;

import org.testcontainers.containers.PostgreSQLContainer;

import org.testcontainers.junit.jupiter.Testcontainers;

@Testcontainers

@ContextConfiguration(initializers = {AuditVizualizationBaseTest.Initializer.class})

@TestExecutionListeners({DependencyInjectionTestExecutionListener.class, FlywayTestExecutionListener.class, SqlScriptsTestExecutionListener.class})

public abstract class AuditVizualizationBaseTest {

static final PostgreSQLContainer POSTGRE\_SQL\_CONTAINER;

static {

POSTGRE\_SQL\_CONTAINER = new PostgreSQLContainer("postgres:15.2")

.withDatabaseName("calculator\_db")

.withUsername("postgres")

.withPassword("1111");

POSTGRE\_SQL\_CONTAINER.start();

Flyway.configure()

.dataSource(POSTGRE\_SQL\_CONTAINER.getJdbcUrl(), POSTGRE\_SQL\_CONTAINER.getUsername(), POSTGRE\_SQL\_CONTAINER.getPassword())

.cleanDisabled(false)

.load()

.migrate();

}

static class Initializer implements ApplicationContextInitializer<ConfigurableApplicationContext> {

public void initialize(ConfigurableApplicationContext configurableApplicationContext) {

TestPropertyValues.of(

"database.cache.url=" + POSTGRE\_SQL\_CONTAINER.getJdbcUrl(),

"database.cache.username=" + POSTGRE\_SQL\_CONTAINER.getUsername(),

"database.cache.password=" + POSTGRE\_SQL\_CONTAINER.getPassword(),

"spring.flyway.url=" + POSTGRE\_SQL\_CONTAINER.getJdbcUrl(),

"spring.flyway.user=" + POSTGRE\_SQL\_CONTAINER.getUsername(),

"spring.flyway.password=" + POSTGRE\_SQL\_CONTAINER.getPassword()

).applyTo(configurableApplicationContext.getEnvironment());

}

}

}