МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

ОТЧЕТ

ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| к. т. н. |  |  |  | Степанов П.А. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

«РЕАКТИВНАЯ АРХИТЕКТУРА»

ПО ДИСЦИПЛИНЕ «ПРОЕКТИРОВАНИЕ СЕРВЕРНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ»

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4030М |  |  |  | Коновалов Л.О. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург

2021

**Задание:**

1. Выбрать предметную область и с использованием технологии WebFlux реализовать нагруженный RESTful API с правильной обработкой статус кодов и тестами.

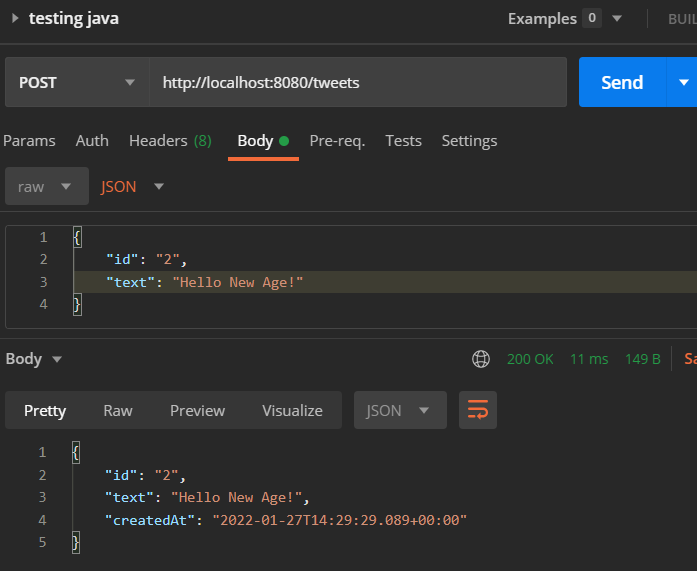
2. Покрыть этот функционал достаточным количеством юнит-тестов.

**Ход работы:**

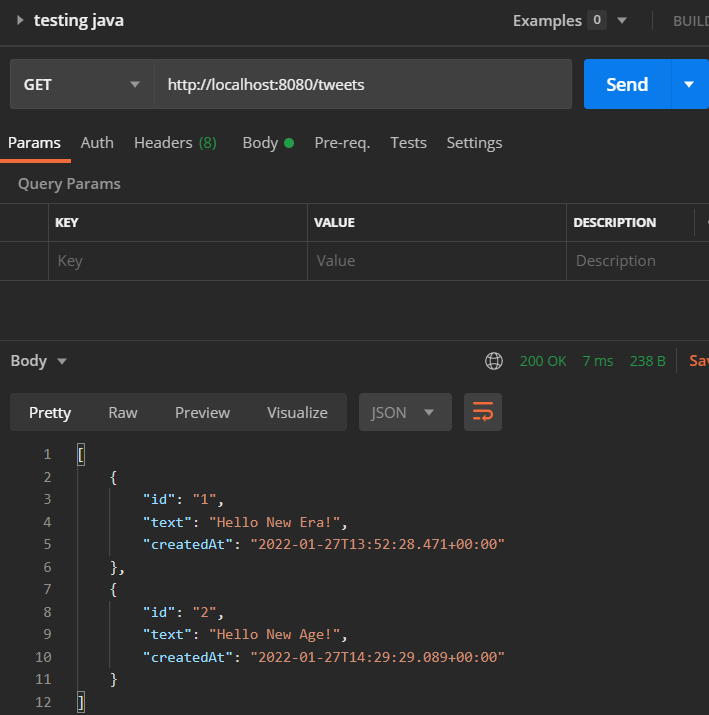
**Предметная область:** приложение для выкладывания небольших сообщений в сеть (аналог твиттера)

В проекте реализованы следующие REST APIs:

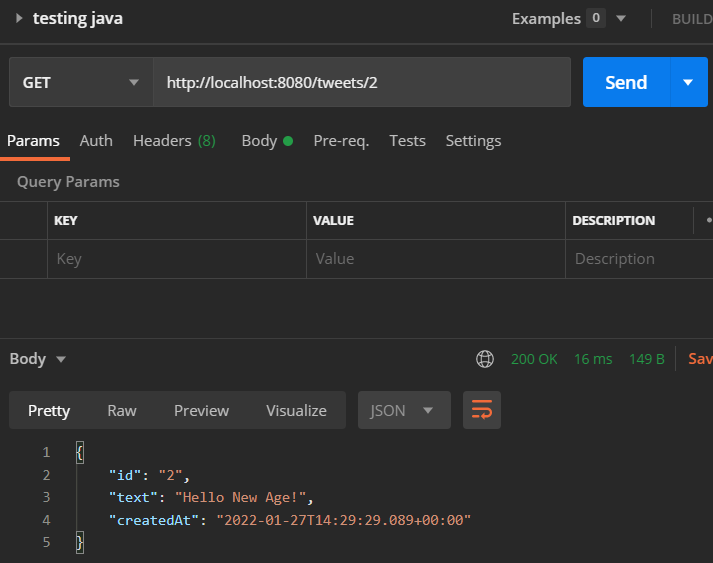
POST:



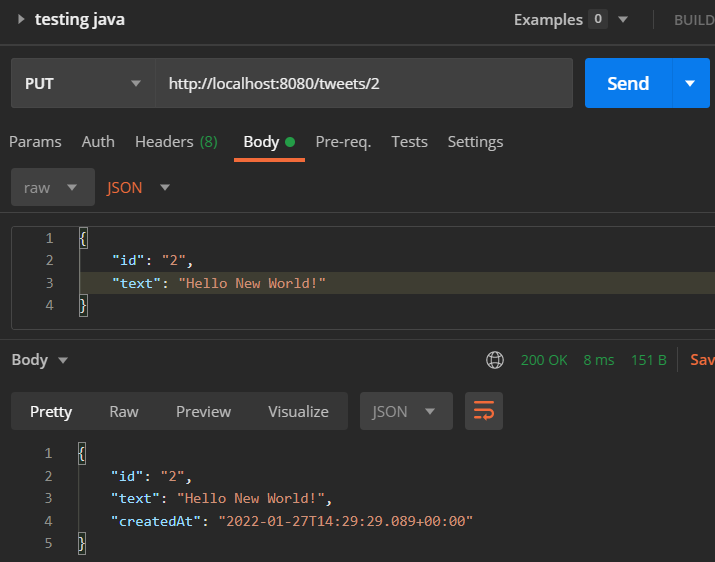
GET:



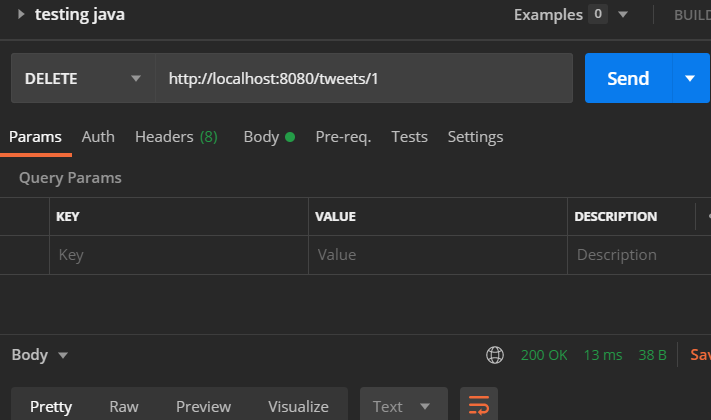
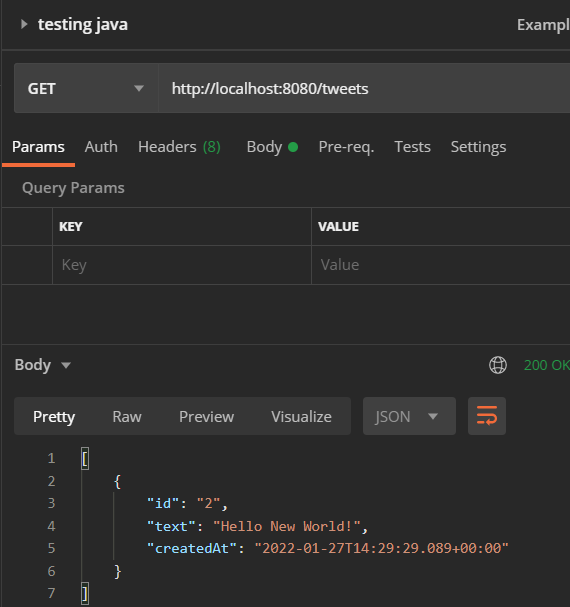
GET по id:



PUT:

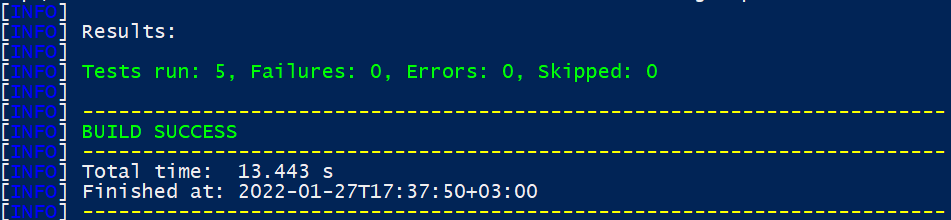


DELETE:

Также данный функционал был покрыт юнит тестами.

Результат работы тестов:



**Вывод:**

В рамках лабораторной работы была реализован нагруженный RESTful API с использованием технологии WebFlux. В качестве предметной области использовалась программа для выкладывания небольших сообщений в сеть. Созданный функционал был покрыт достаточным количеством юнит-тестов для проверки работоспособности и корректной обработки входящих команд.

Приложение

Тесты:

package com.lr.webfluxrest;

import com.lr.webfluxrest.model.Tweet;

import com.lr.webfluxrest.repository.TweetRepository;

import org.assertj.core.api.Assertions;

import org.junit.jupiter.api.Test;

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

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

import org.springframework.http.MediaType;

import org.springframework.test.web.reactive.server.WebTestClient;

import reactor.core.publisher.Mono;

import java.util.Collections;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)

public class WebfluxrestApplicationTests {

    @Autowired

    private WebTestClient webTestClient;

    @Autowired

    TweetRepository tweetRepository;

    @Test

    public void testCreateTweet() {

        Tweet tweet = new Tweet("This is a Test Tweet");

        webTestClient.post().uri("/tweets")

                .contentType(MediaType.APPLICATION\_JSON)

                .accept(MediaType.APPLICATION\_JSON)

                .body(Mono.just(tweet), Tweet.class)

                .exchange()

                .expectStatus().isOk()

                .expectHeader().contentType(MediaType.APPLICATION\_JSON)

                .expectBody()

                .jsonPath("$.id").isNotEmpty()

                .jsonPath("$.text").isEqualTo("This is a Test Tweet");

    }

    @Test

    public void testGetAllTweets() {

        webTestClient.get().uri("/tweets")

                .accept(MediaType.APPLICATION\_JSON)

                .exchange()

                .expectStatus().isOk()

                .expectHeader().contentType(MediaType.APPLICATION\_JSON)

                .expectBodyList(Tweet.class);

    }

    @Test

    public void testGetSingleTweet() {

        Tweet tweet = tweetRepository.save(new Tweet("Hello, World!")).block();

        webTestClient.get()

                .uri("/tweets/{id}", Collections.singletonMap("id", tweet.getId()))

                .exchange()

                .expectStatus().isOk()

                .expectBody()

                .consumeWith(response ->

                        Assertions.assertThat(response.getResponseBody()).isNotNull());

    }

    @Test

    public void testUpdateTweet() {

        Tweet tweet = tweetRepository.save(new Tweet("Initial Tweet")).block();

        Tweet newTweetData = new Tweet("Updated Tweet");

        webTestClient.put()

                .uri("/tweets/{id}", Collections.singletonMap("id", tweet.getId()))

                .contentType(MediaType.APPLICATION\_JSON)

                .accept(MediaType.APPLICATION\_JSON)

                .body(Mono.just(newTweetData), Tweet.class)

                .exchange()

                .expectStatus().isOk()

                .expectHeader().contentType(MediaType.APPLICATION\_JSON)

                .expectBody()

                .jsonPath("$.text").isEqualTo("Updated Tweet");

    }

    @Test

    public void testDeleteTweet() {

        Tweet tweet = tweetRepository.save(new Tweet("To be deleted")).block();

        webTestClient.delete()

                .uri("/tweets/{id}", Collections.singletonMap("id",  tweet.getId()))

                .exchange()

                .expectStatus().isOk();

    }

}

Controller:

package com.lr.webfluxrest.controller;

import com.lr.webfluxrest.exception.TweetNotFoundException;

import com.lr.webfluxrest.model.Tweet;

import com.lr.webfluxrest.payload.ErrorResponse;

import com.lr.webfluxrest.repository.TweetRepository;

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

import org.springframework.dao.DuplicateKeyException;

import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import reactor.core.publisher.Flux;

import reactor.core.publisher.Mono;

import javax.validation.Valid;

@RestController

public class TweetController {

    @Autowired

    private TweetRepository tweetRepository;

    @GetMapping("/tweets")

    public Flux<Tweet> getAllTweets() {

        return tweetRepository.findAll();

    }

    @PostMapping("/tweets")

    public Mono<Tweet> createTweets(@Valid @RequestBody Tweet tweet) {

        return tweetRepository.save(tweet);

    }

    @GetMapping("/tweets/{id}")

    public Mono<ResponseEntity<Tweet>> getTweetById(@PathVariable(value = "id") String tweetId) {

        return tweetRepository.findById(tweetId)

                .map(savedTweet -> ResponseEntity.ok(savedTweet))

                .defaultIfEmpty(ResponseEntity.notFound().build());

    }

    @PutMapping("/tweets/{id}")

    public Mono<ResponseEntity<Tweet>> updateTweet(@PathVariable(value = "id") String tweetId,

                                                   @Valid @RequestBody Tweet tweet) {

        return tweetRepository.findById(tweetId)

                .flatMap(existingTweet -> {

                    existingTweet.setText(tweet.getText());

                    return tweetRepository.save(existingTweet);

                })

                .map(updateTweet -> new ResponseEntity<>(updateTweet, HttpStatus.OK))

                .defaultIfEmpty(new ResponseEntity<>(HttpStatus.NOT\_FOUND));

    }

    @DeleteMapping("/tweets/{id}")

    public Mono<ResponseEntity<Void>> deleteTweet(@PathVariable(value = "id") String tweetId) {

        return tweetRepository.findById(tweetId)

                .flatMap(existingTweet ->

                        tweetRepository.delete(existingTweet)

                            .then(Mono.just(new ResponseEntity<Void>(HttpStatus.OK)))

                )

                .defaultIfEmpty(new ResponseEntity<>(HttpStatus.NOT\_FOUND));

    }

    @ExceptionHandler(DuplicateKeyException.class)

    public ResponseEntity handleDuplicateKeyException(DuplicateKeyException ex) {

        return ResponseEntity.status(HttpStatus.CONFLICT).body(new ErrorResponse("A Tweet with the same text already exists"));

    }

    @ExceptionHandler(TweetNotFoundException.class)

    public ResponseEntity handleTweetNotFoundException(TweetNotFoundException ex) {

        return ResponseEntity.notFound().build();

    }

}

Пример обработки ошибки отсутствия твита при запросе:

package com.lr.webfluxrest.exception;

public class TweetNotFoundException extends RuntimeException {

    public TweetNotFoundException(String tweetId) {

        super("Tweet not found with id " + tweetId);

    }

}

Model:

package com.lr.webfluxrest.model;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import javax.validation.constraints.NotBlank;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

import java.util.Date;

@Document(collection = "tweets")

public class Tweet {

    @Id

    private String id;

    @NotBlank

    @Size(max = 140)

    private String text;

    @NotNull

    private Date createdAt = new Date();

    public Tweet() {

    }

    public Tweet(String text) {

        this.id = id;

        this.text = text;

    }

    public String getId() {

        return id;

    }

    public void setId(String id) {

        this.id = id;

    }

    public String getText() {

        return text;

    }

    public void setText(String text) {

        this.text = text;

    }

    public Date getCreatedAt() {

        return createdAt;

    }

    public void setCreatedAt(Date createdAt) {

        this.createdAt = createdAt;

    }

}