Лабораторная работа №1

Код программы продемонстрирован на рисунках 1,2,3:

1) Класс **CountController**:

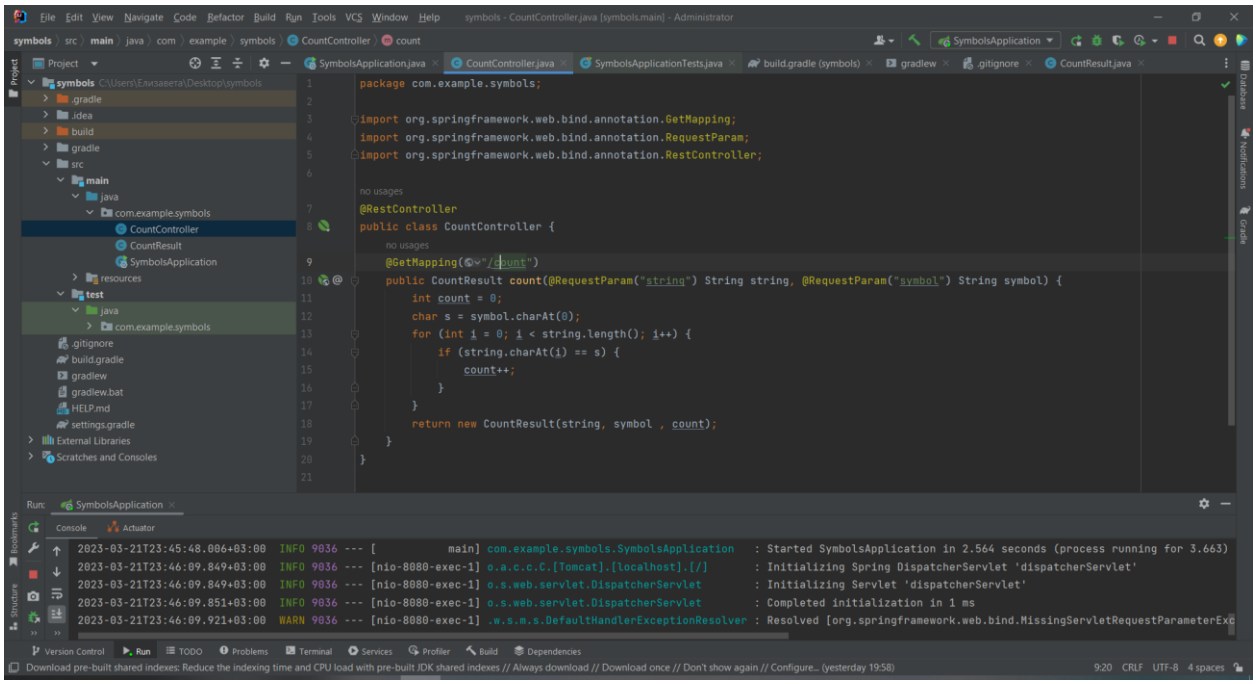


Рисунок 1

2) Класс **CountResult**:

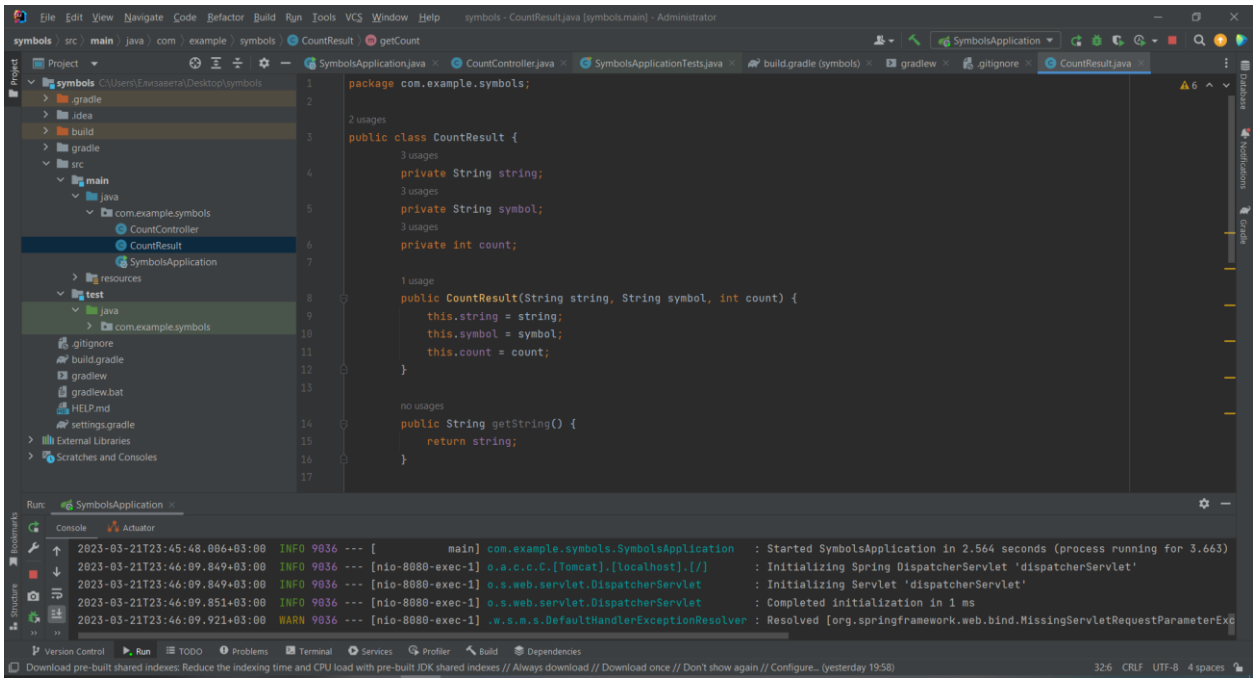


Рисунок 2

Продолжение данного класса:

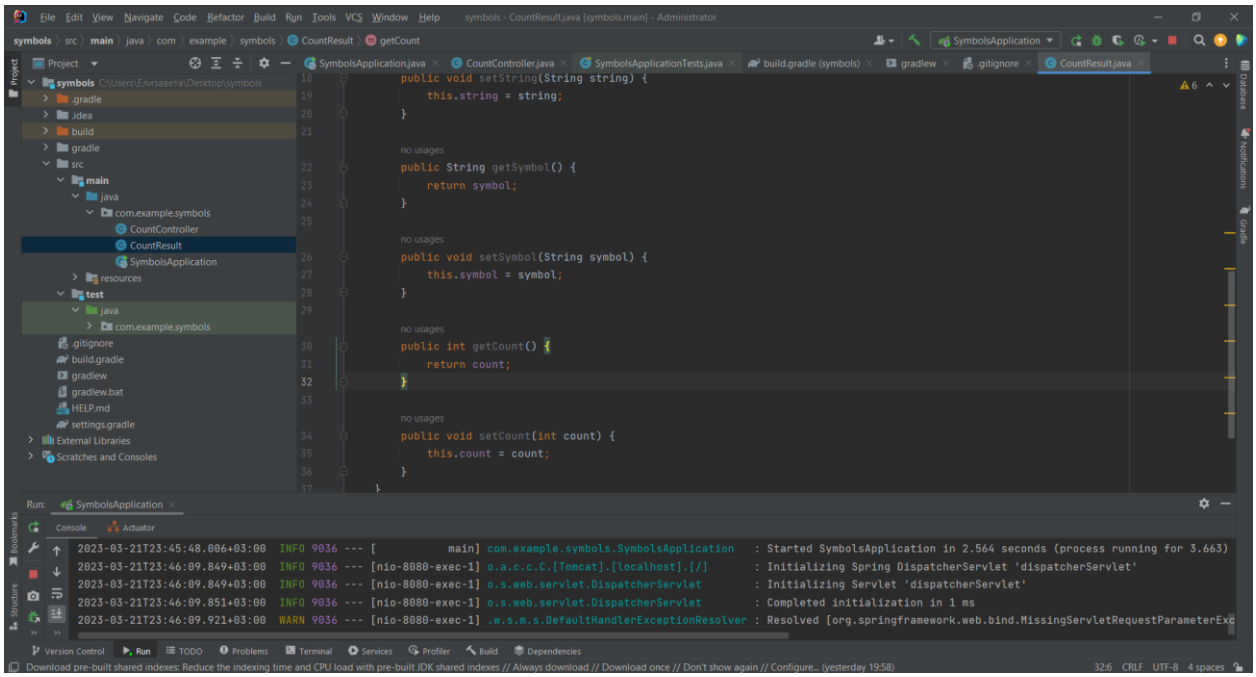


Рисунок 3

После запуска приложения, переходя по адресу http://localhost:8080/count?string=yourstring&symbol=y в веб-браузере, где yourstring - это строка, в которой по заданию нужно найти символ, а y - символ, который будем искать и подсчитывать, сколько раз он встречается в данной строке. Далее видим ответ в формате JSON в браузере.

Скриншоты, на которых продемонстрирована работа программы (см. рис. 4,5,6,7):

Рисунок 4

Рисунок 5

Рисунок 6

Рисунок 7

Работа программы через Postman (см. рис.8):

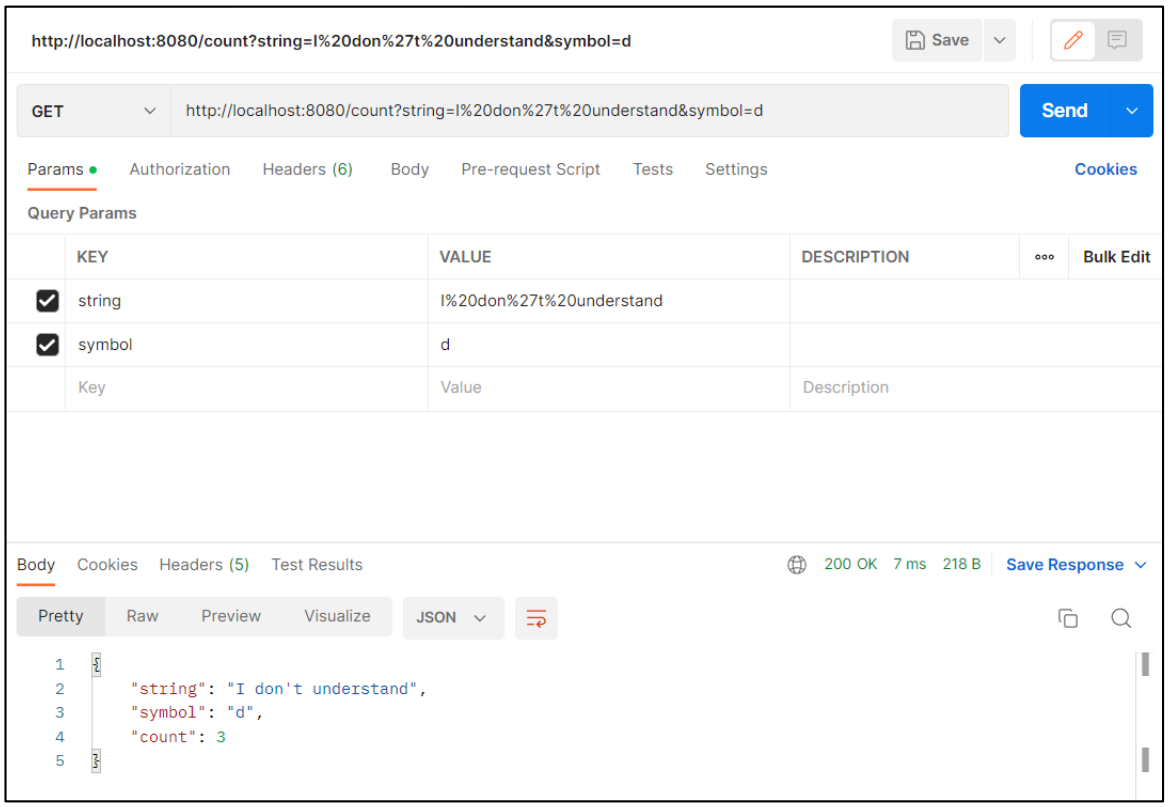


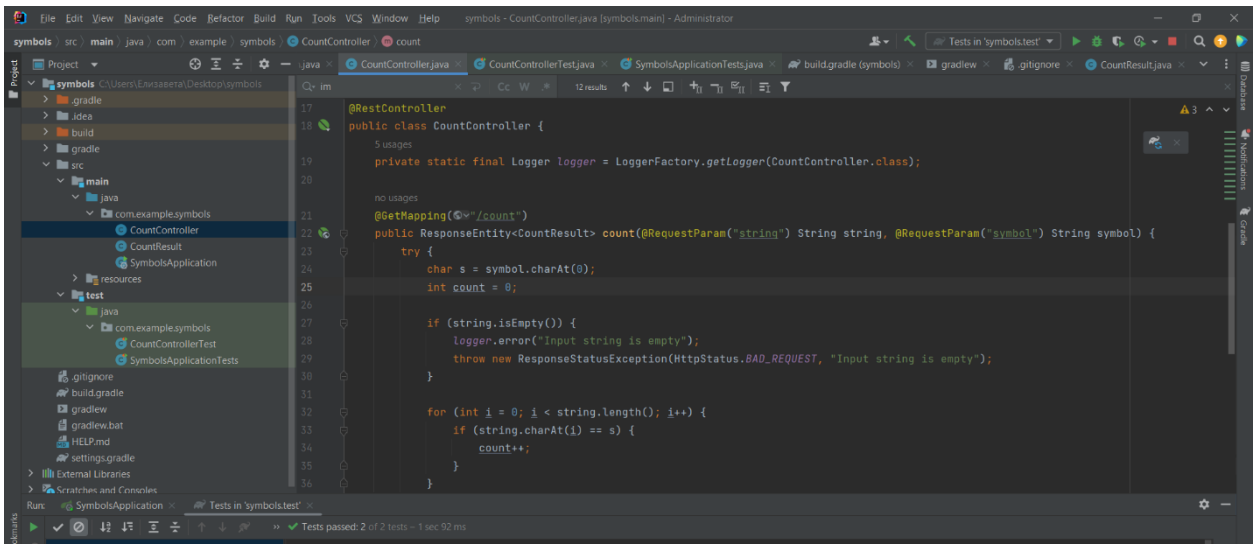
Рисунок 8

Лабораторная работа №2

В данном коде были добавлены:

- Валидация входных параметров с возвращением 400 ошибки;
- Обработка внутренних unchecked ошибок с возвратом 500 ошибки;
- Логирование действий и ошибок

Код изображён на рисунках 9,10



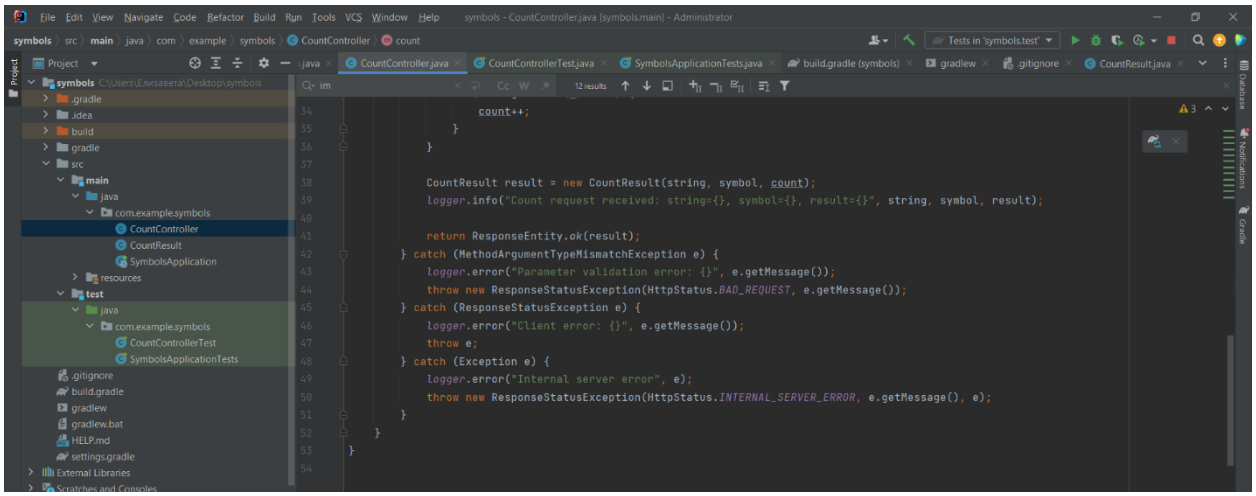
```
@RestController
public class CountController {
    5 usages
    private static final Logger logger = LoggerFactory.getLogger(CountController.class);

    no usages
    @GetMapping("/count")
    public ResponseEntity<CountResult> count(@RequestParam("string") String string, @RequestParam("symbol") String symbol) {
        try {
            char s = symbol.charAt(0);
            int count = 0;

            if (string.isEmpty()) {
                logger.error("Input string is empty");
                throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Input string is empty");
            }

            for (int i = 0; i < string.length(); i++) {
                if (string.charAt(i) == s) {
                    count++;
                }
            }
```

Рисунок 9



```
                    count++;
                }
            }

            CountResult result = new CountResult(string, symbol, count);
            logger.info("Count request received: string={}, symbol={}, result={}", string, symbol, result);

            return ResponseEntity.ok(result);
        } catch (MethodArgumentTypeMismatchException e) {
            logger.error("Parameter validation error: {}", e.getMessage());
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
        } catch (ResponseStatusException e) {
            logger.error("Client error: {}", e.getMessage());
            throw e;
        } catch (Exception e) {
            logger.error("Internal server error", e);
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, e.getMessage(), e);
        }
    }
}
```

Рисунок 10

Скриншоты, на которых продемонстрирована работа программы (см. рис. 11,12,13,14):

400 ошибка:

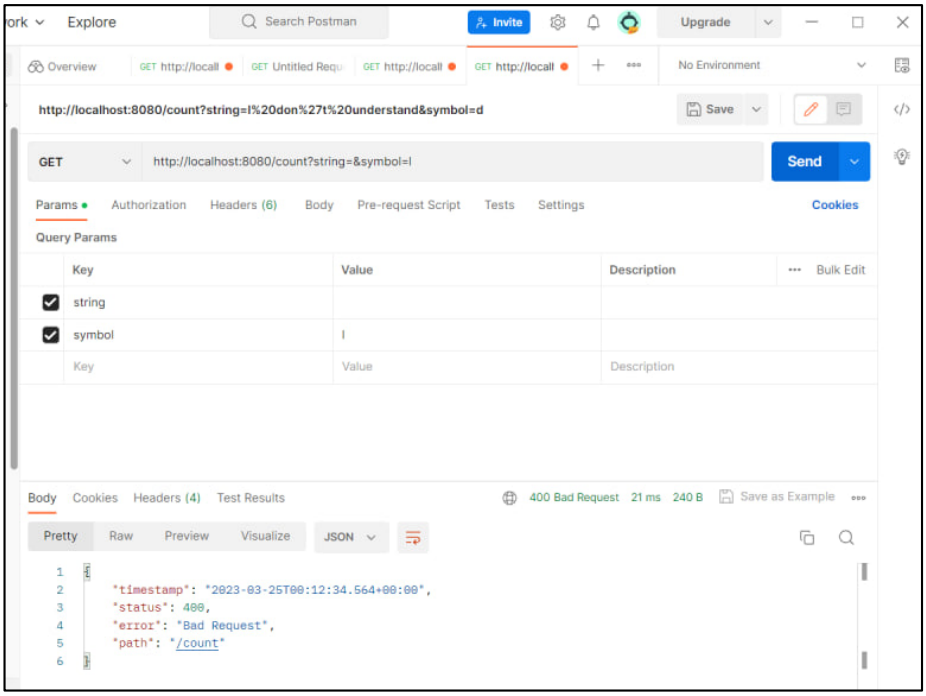


Рисунок 11

Рисунок 12

500 ошибка:

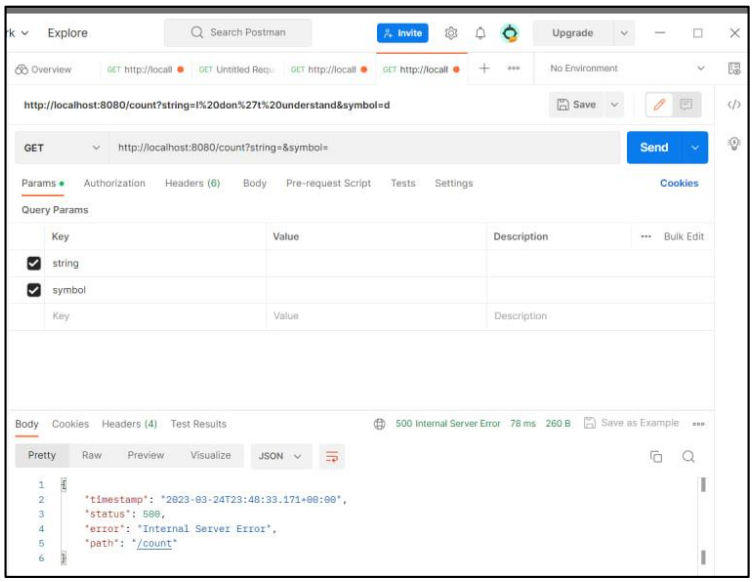


Рисунок 13

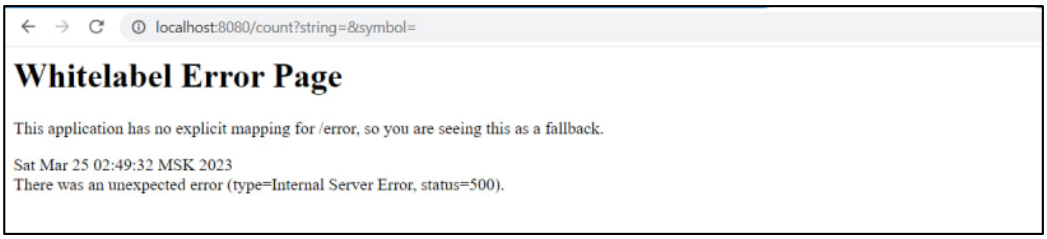


Рисунок 14

Далее отображено выполнение задания: написать unit test. См. рис.15,16,17.

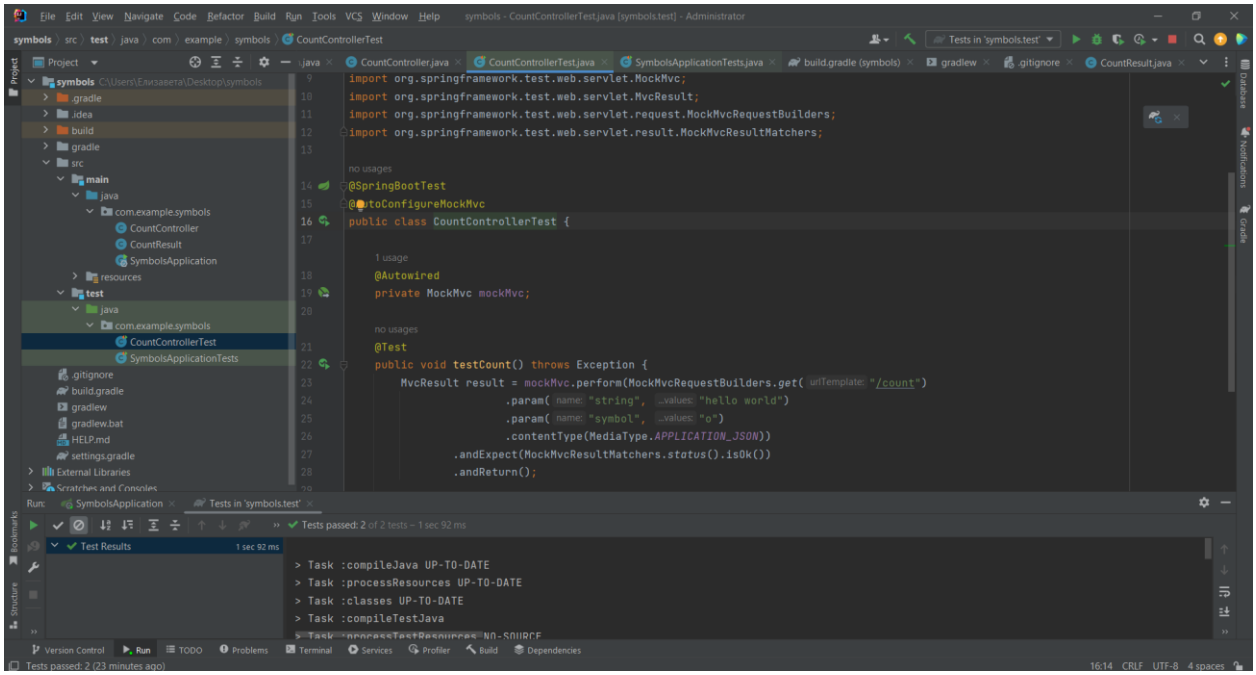


Рисунок 15

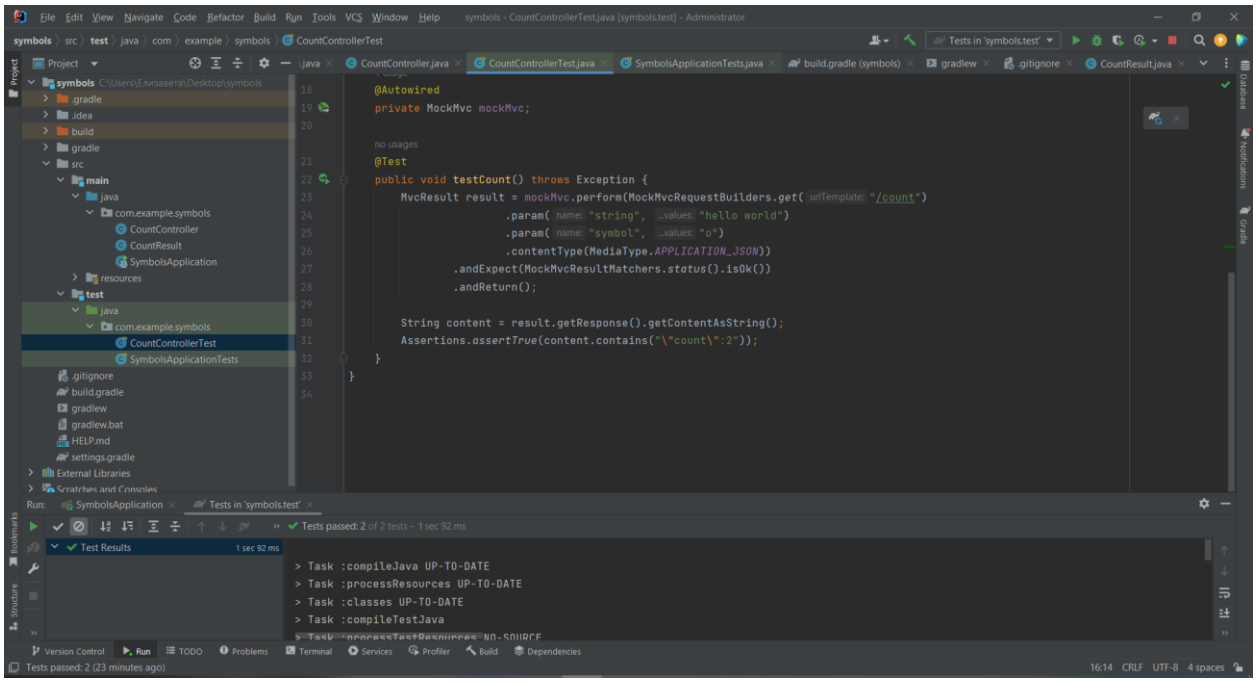


Рисунок 16

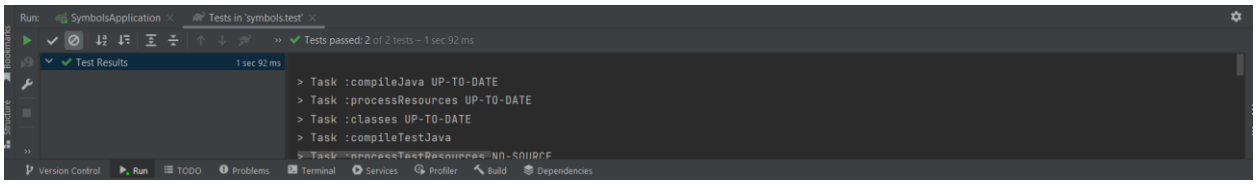

Run:
SymbolsApplication
Tests in 'symbols.test'
Tests passed: 2 of 2 tests – 1 sec 92 ms
Test Results
1 sec 92 ms
> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava
Version Control
Run
TODO
Problems
Terminal
Services
Profiler
Build
Dependencies

Рисунок 17

Лабораторная работа №3

1) Класс **CountController** (см. рис.18, 19,20,21)**:**

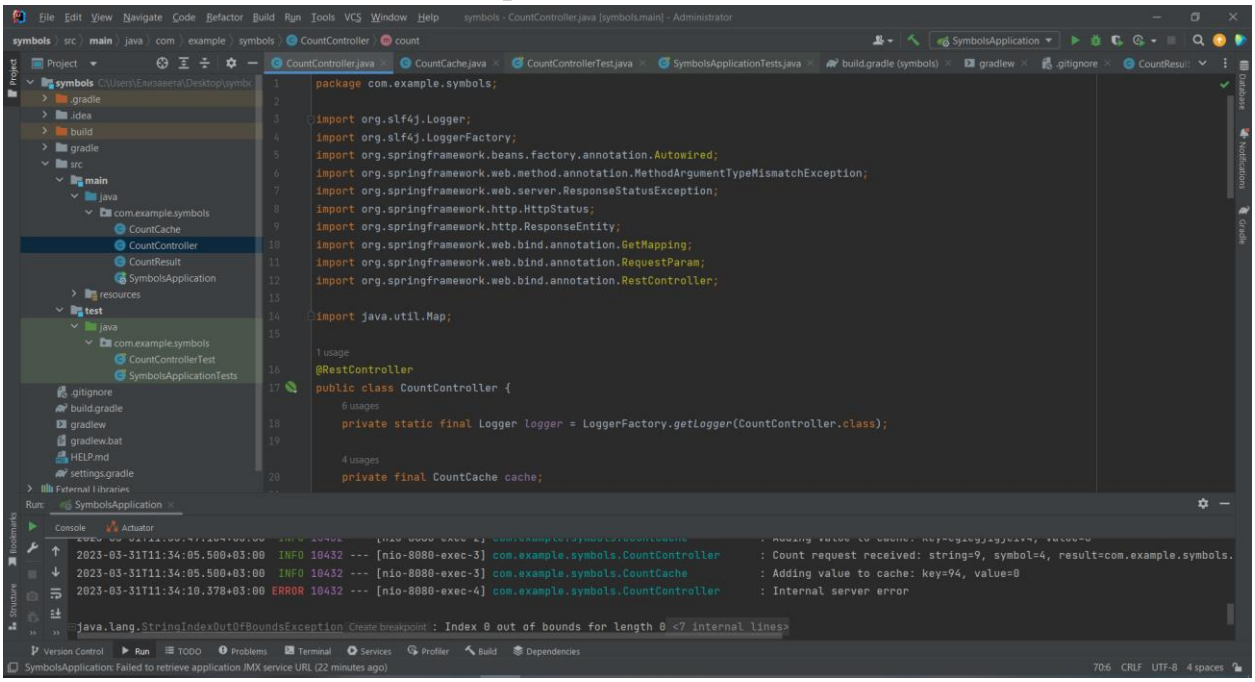


Рисунок 18

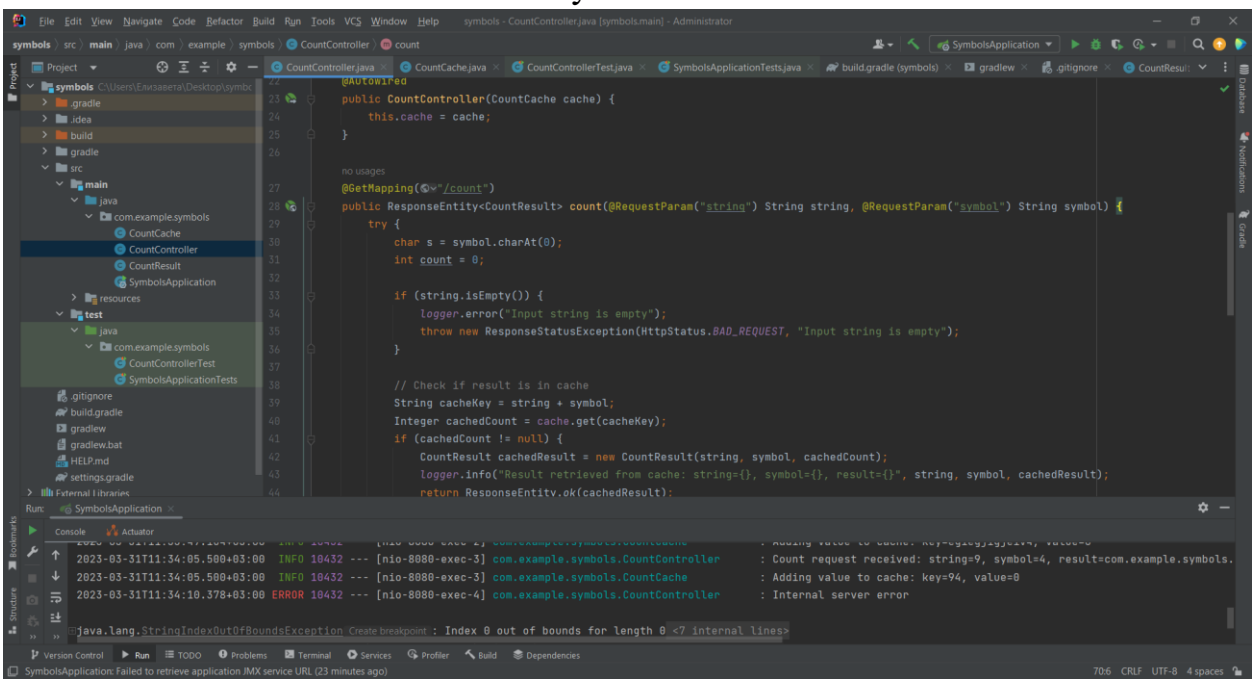


Рисунок 19

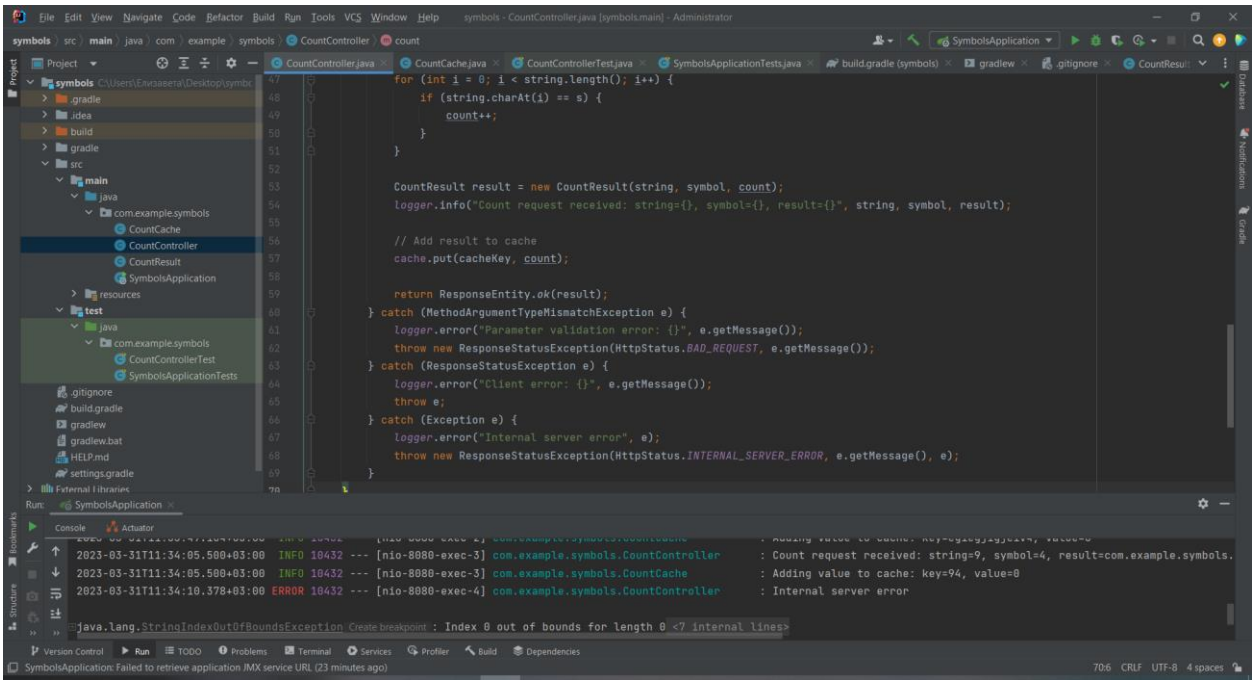


Рисунок 20

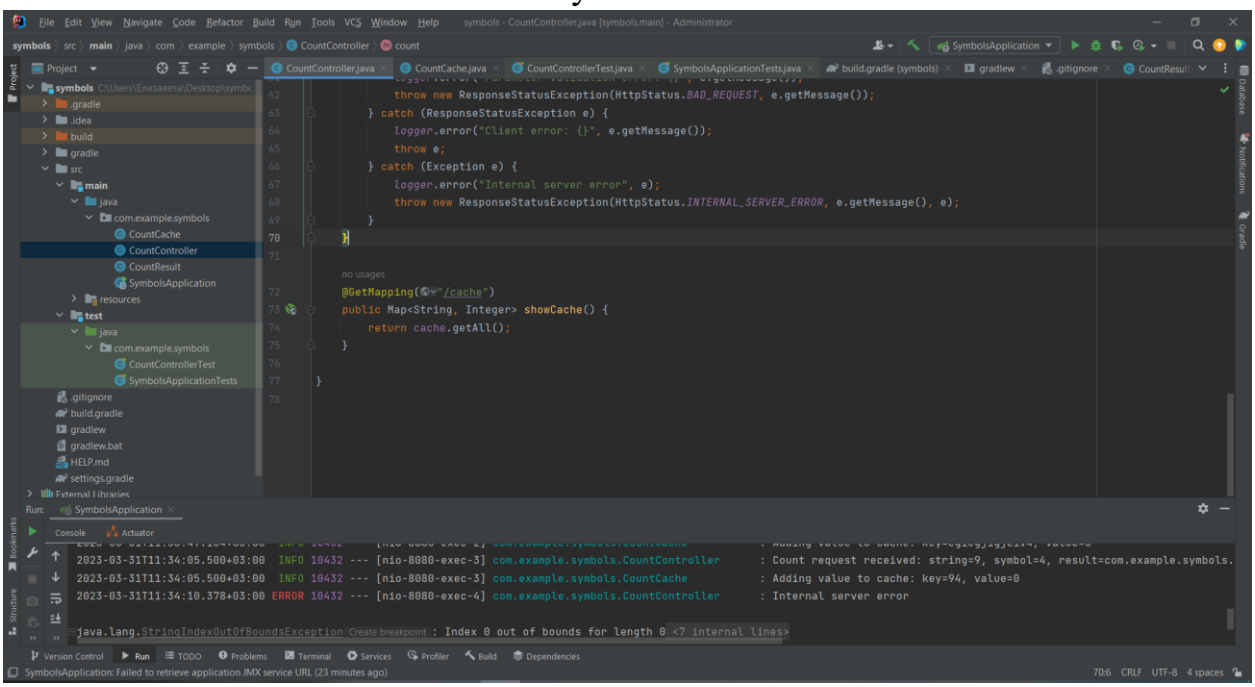


Рисунок 21

2) Класс CountCache (см.рис. 22,23)**:**

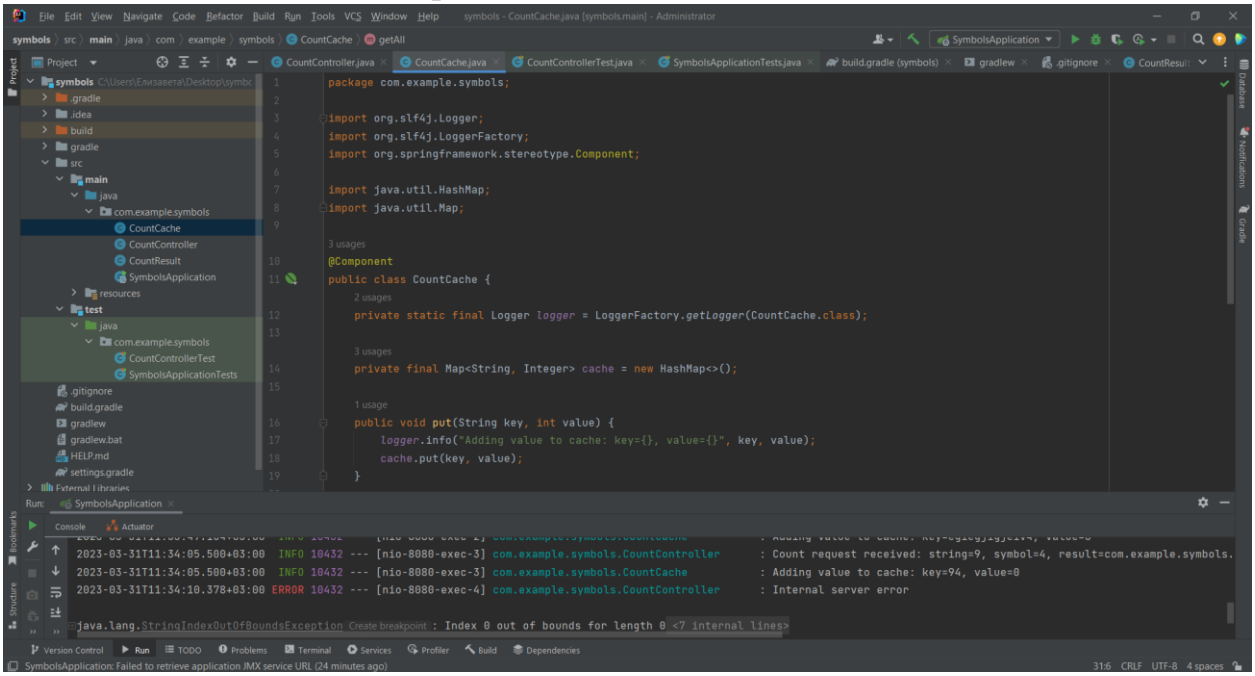


Рисунок 22

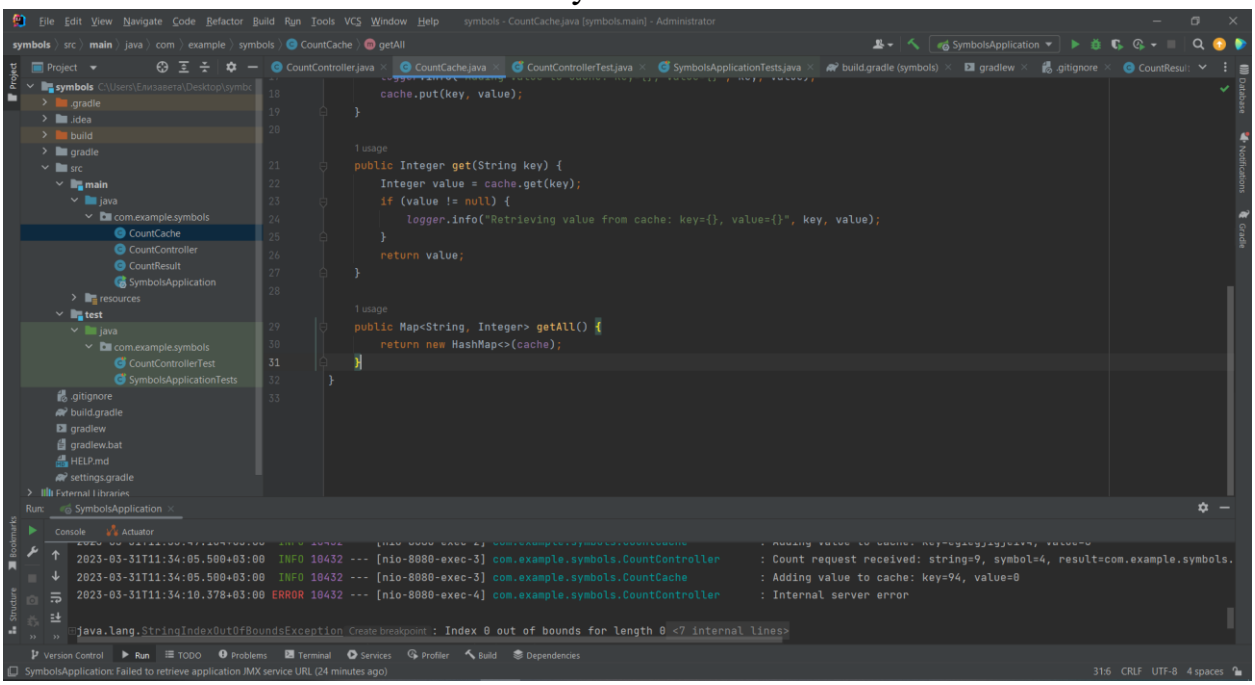


Рисунок 23

Класс CountResult остаётся без изменений.

После того, как произойдёт переход по URL /cache в браузере, Postman получим (см.рис.24,25):

- Если 3:

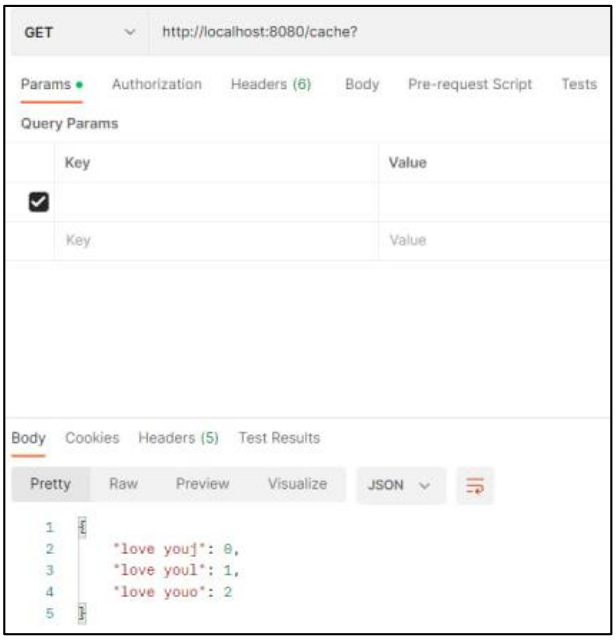


Рисунок 24

- Если 2:

Рисунок 25