

СОДЕРЖАНИЕ

1. ПРАКТИЧЕСКАЯ РАБОТА № 2.....	2
1. 1 Выполнение задания по варианту	2
1. 2 Ответы на вопросы	11
ВЫВОД ПО ПРАКТИЧЕСКОЙ №2.....	12

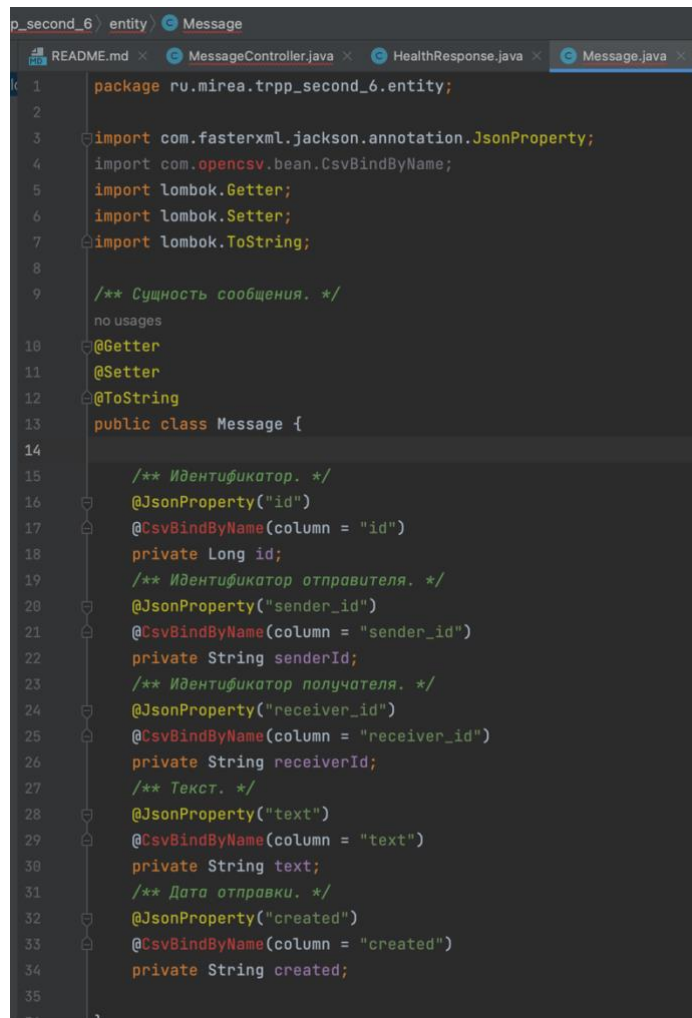
1. ПРАКТИЧЕСКАЯ РАБОТА № 2

1. 1 Выполнение задания по варианту №6

Задание выполнено согласно шестому варианту. Был взят репозиторий: <https://github.com/rtu-mirea/trpp-second-6>, сущность

ru.mirea.entity.Message. По заданию был клонирован git-репозиторий.

На рисунках 1 и 2 представлены исправленные ранее отсутствующие зависимости в соответствующих блоках в build.gradle, чтобы проект снова начал собираться.



```
1 package ru.mirea.trpp_second_6.entity;
2
3 import com.fasterxml.jackson.annotation.JsonProperty;
4 import com.opencsv.bean.CsvBindByName;
5 import lombok.Getter;
6 import lombok.Setter;
7 import lombok.ToString;
8
9 /** Сущность сообщения. */
10 no usages
11 @Getter
12 @Setter
13 @ToString
14 public class Message {
15
16     /** Идентификатор. */
17     @JsonProperty("id")
18     @CsvBindByName(column = "id")
19     private Long id;
20     /** Идентификатор отправителя. */
21     @JsonProperty("sender_id")
22     @CsvBindByName(column = "sender_id")
23     private String senderId;
24     /** Идентификатор получателя. */
25     @JsonProperty("receiver_id")
26     @CsvBindByName(column = "receiver_id")
27     private String receiverId;
28     /** Текст. */
29     @JsonProperty("text")
30     @CsvBindByName(column = "text")
31     private String text;
32     /** Дата отправки. */
33     @JsonProperty("created")
34     @CsvBindByName(column = "created")
35     private String created;
```

Рисунок 1 – отсутствующая зависимость Message.java

```
dependencies {
    annotationProcessor 'org.projectlombok:lombok:1.18.18'
    compileOnly 'org.projectlombok:lombok:1.18.18'

    implementation("com.opencsv:opencsv:5.7.1")

    implementation("io.micronaut:micronaut-runtime")
    implementation("io.micronaut:micronaut-validation")
    implementation("io.micronaut:micronaut-http-client")
    implementation("javax.annotation:javax.annotation-api")
    implementation("org.apache.logging.log4j:log4j-core:2.12.1")
    runtimeOnly("org.apache.logging.log4j:log4j-api:2.12.1")
    runtimeOnly("org.apache.logging.log4j:log4j-slf4j-impl:2.12.1")
}
```

Рисунок 2 - исправленная зависимость в файле с разрешением gradle

На рисунке 3 показано, что отсутствующие зависимости устранены.

```
package ru.mirea.trpp_second_6.entity;

import com.fasterxml.jackson.annotation.JsonProperty;
import com.opencsv.bean.CsvBindByName;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

/** Сущность сообщения. */
no usages
@Getter
@Setter
@ToString
public class Message {

    /** Идентификатор. */
    @JsonProperty("id")
    @CsvBindByName(column = "id")
    private Long id;

    /** Идентификатор отправителя. */
    @JsonProperty("sender_id")
    @CsvBindByName(column = "sender_id")
    private String senderId;

    /** Идентификатор получателя. */
    @JsonProperty("receiver_id")
    @CsvBindByName(column = "receiver_id")
    private String receiverId;

    /** Текст. */
    @JsonProperty("text")
    @CsvBindByName(column = "text")
    private String text;

    /** Дата отправки. */
    @JsonProperty("created")
    @CsvBindByName(column = "created")
    private String created;
}
```

Рисунок 3 - исправленная зависимость в Message.java

На рисунке 4 показано, что в классе MessageController отсутствовал подключение класса Movie. На рисунке 5 представлен его импорт.

```
package ru.mirea.trpp_second_6.controllers;

import com.opencsv.bean.CsvToBeanBuilder;
import io.micronaut.http.HttpResponse;
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;

import java.io.InputStreamReader;
import java.util.List;
import java.util.Optional;

/** Контроллер для работы с сообщениями. */
no usages
@Controller("/message")
public class MessageController {

    /** Список сообщений. */
    3 usages
    private final List<Message> messageList;

    /** Конструктор. */
    no usages
    public MessageController() {
        messageList = new CsvToBeanBuilder<Message>(new InputStreamReader(this.getClass().getResourceAsStream("message.csv")))
    }

    /**
     * Получить список сообщений.
     * @return список сообщений
     */
    no usages
    @Get()
    public HttpResponse<List<Message>> getMessages() { return HttpResponse.ok(messageList); }

    /**
     * Найти сообщение по идентификатору.

```

Рисунок 4 – отсутствие подключения класса Message в MessageController.java

```
second_6 controllers MessageController
README.md MessageController.java HealthResponse.java Message.java build.gradle (trpp-second-6) TestTest.java

1 import com.opencsv.bean.CsvToBeanBuilder;
2 import io.micronaut.http.HttpResponse;
3 import io.micronaut.http.annotation.Controller;
4 import io.micronaut.http.annotation.Get;
5
6 import ru.mirea.trpp_second_6.entity.Message;
7
8
9 import java.io.InputStreamReader;
10 import java.util.List;
11 import java.util.Optional;
12
13 /** Контроллер для работы с сообщениями. */
14 no usages
15 @Controller("/message")
16 public class MessageController {
17
18     /** Список сообщений. */
19     3 usages
20     private final List<Message> messageList;
21
22     /** Конструктор. */
23     no usages
24     public MessageController() {
25         messageList = new CsvToBeanBuilder<Message>(new InputStreamReader(this.getClass().getResourceAsStream("message.csv")))
26     }
27
28     /**
29      * Получить список сообщений.
30      * @return список сообщений
31      */
32     no usages
33     @Get()
34     public HttpResponse<List<Message>> getMessages() { return HttpResponse.ok(messageList); }
35
36     /**

```

Рисунок 5 - import класса Message

Необходимо по заданию собрать документацию проекта, найти в ней запросы состояния и сущности по идентификатору. На рисунке 6 представлен процесс создания документация, а на рисунке 7 представлена уже созданная документация, на рисунке 8 показан запрос состояния и на рисунках 9.1, 9.2 представлен запрос сущности по идентификатору.

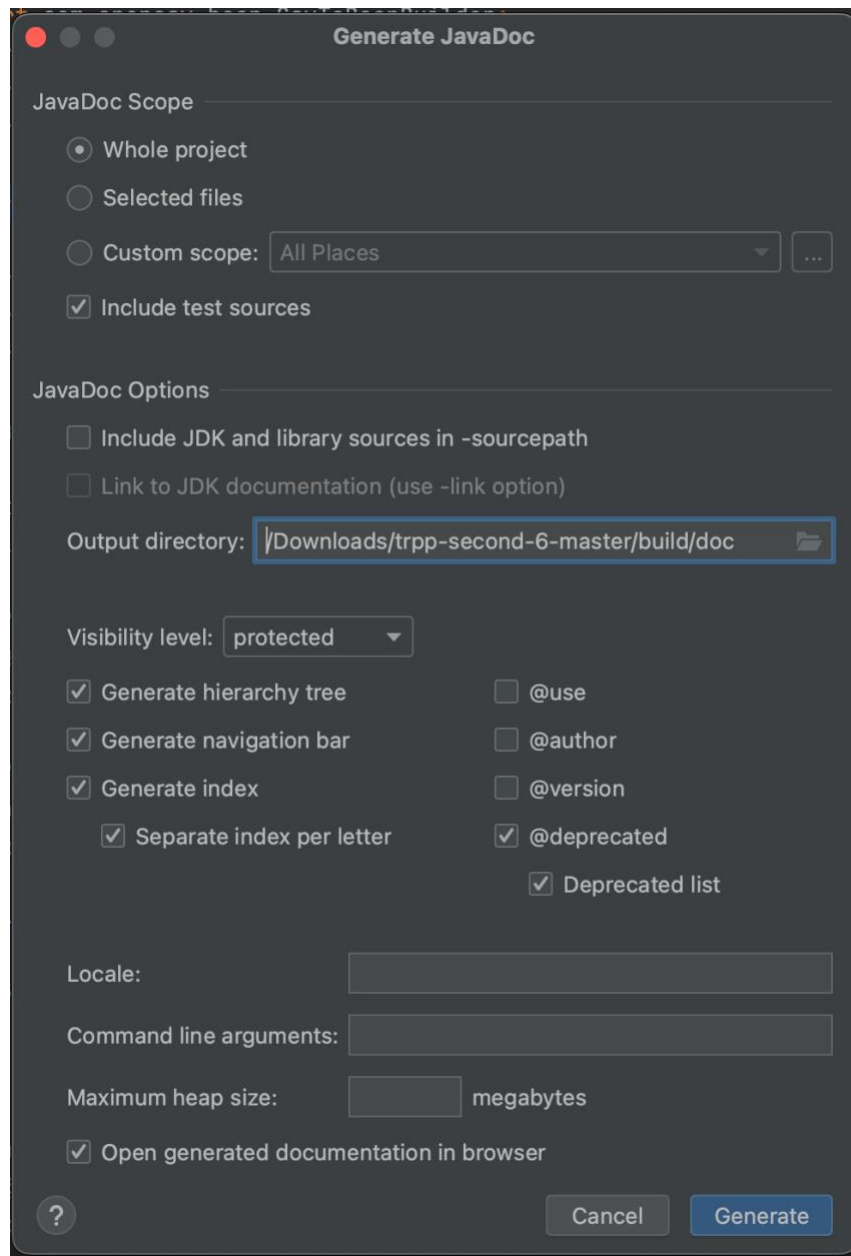


Рисунок 6 – процесс создания документации

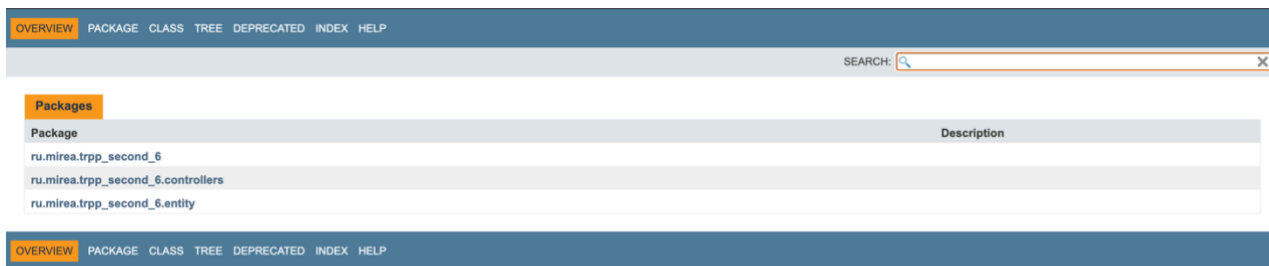


Рисунок 7 – собранная документация

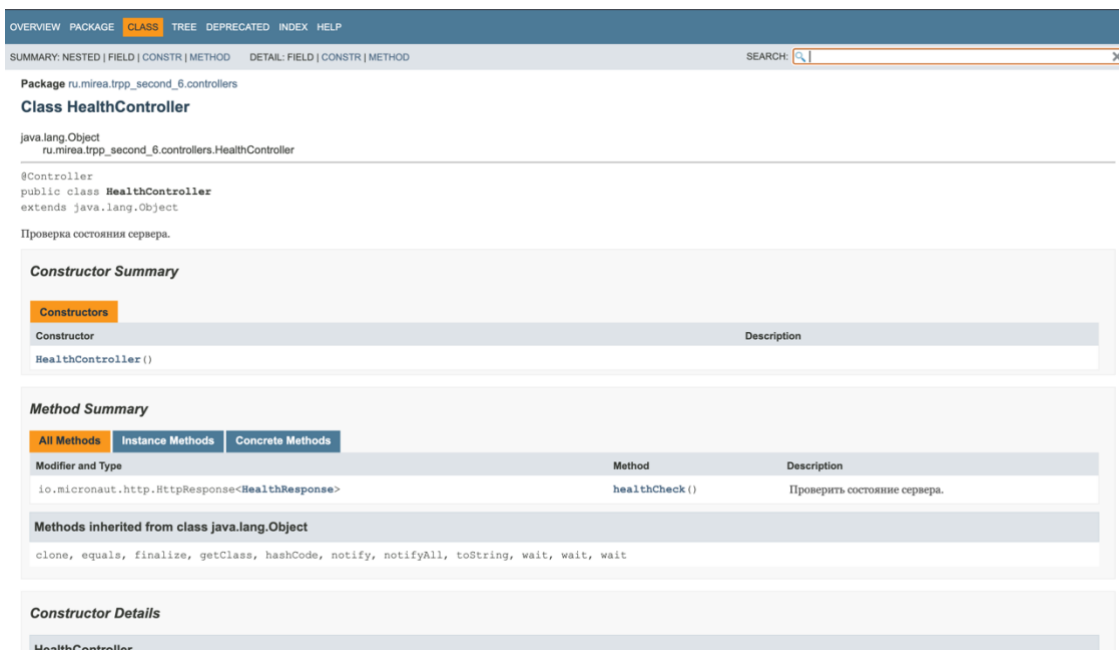


Рисунок 8 – запрос состояния

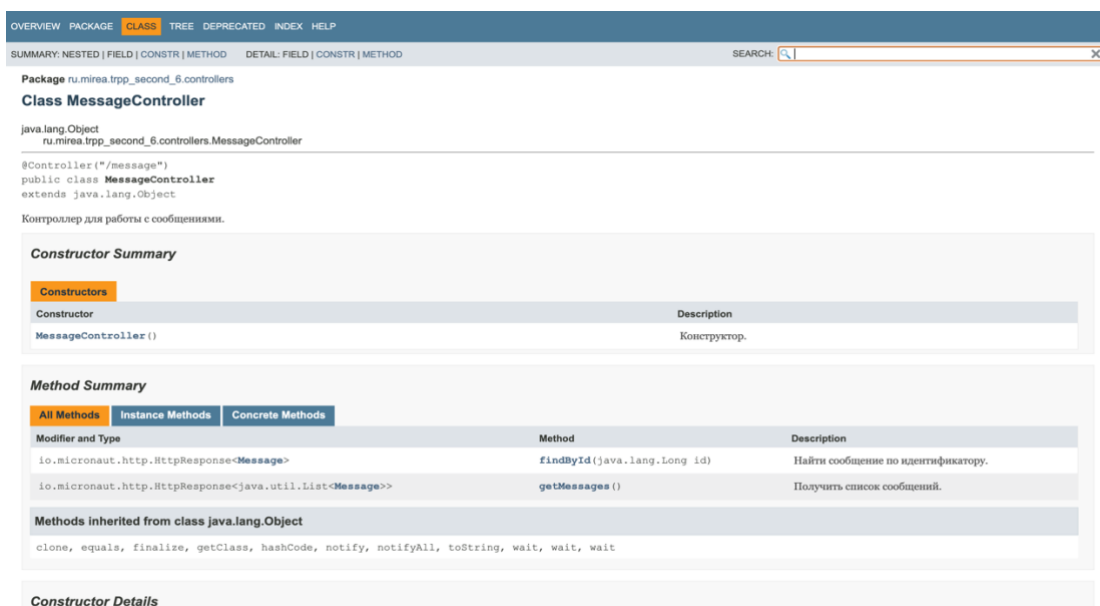


Рисунок 9.1 – запрос сущности по идентификатору

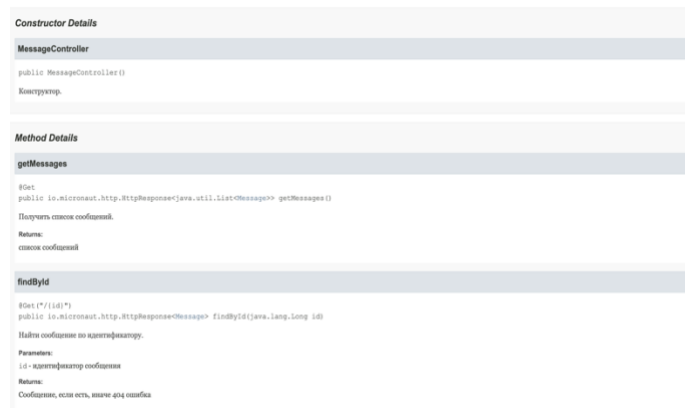


Рисунок 9.2 – запрос сущности по идентификатору

Далее по заданию необходимо собрать файл с расширением jar со всеми зависимостями (так называемый UberJar), после чего запустить приложение. По умолчанию, сервер стартует на порту 8080. На рисунке 10 представлен процесс сборки UberJar (при помощи Gradle task - shadowJar), на рисунке 11 показан jar-файл, а на рисунке 12 представлен запуск приложения.

```

66 ▶ shadowJar {
67     archivesBaseName = "${project.name}"
68     libsDirName = "${project.name}"
69     classifier('')
70 }
71
dependencies{}

```

```

Successful At 19.05 537 ms 22:43:10: Executing 'shadowJar'...

> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :shadowJar UP-TO-DATE

BUILD SUCCESSFUL in 342ms
3 actionable tasks: 3 up-to-date
22:43:10: Execution finished 'shadowJar'.

```

Рисунок 10 – процесс сборки UberJar

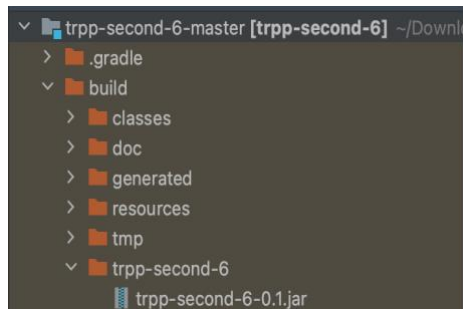


Рисунок 11 – созданный файл с расширением jar

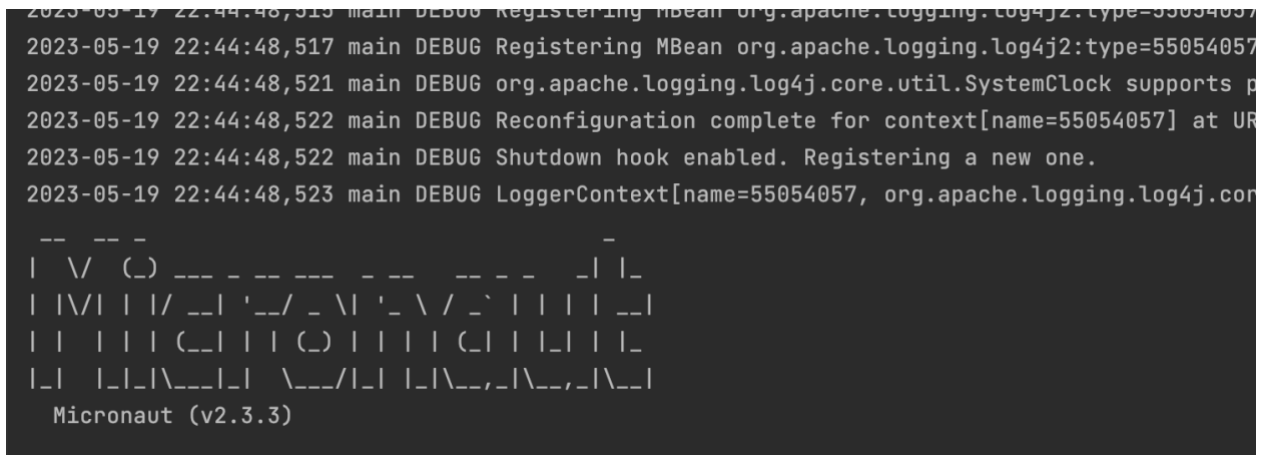


Рисунок 12 – запуск приложения

Теперь нужно запросить состояние запущенного сервера (GET запрос по адресу <http://localhost:8080>).

GET запрос состояния запущенного сервера был отправлен при помощи платформы Postman: <https://www.postman.com/>, для тестирования сервера. На рисунке 13 представлен сам запрос и его результат.

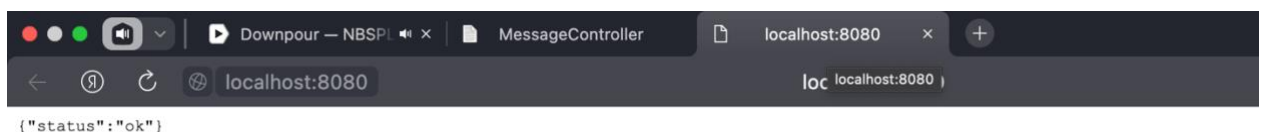


Рисунок 13 – GET запрос состояния запущенного сервера

Далее по заданию необходимо запросить сущность по идентификатору (GET запрос по адресу: <http://localhost:8080/сущность/идентификатор>). Идентификатором будут 3 последних цифры в серийном номере вашего студенческого билета. В данном случае <http://localhost:8080/message/452>.

На рисунке 14 представлен GET запрос сущности по идентификатору.

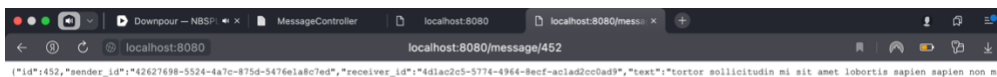


Рисунок 14 – GET запрос сущности по идентификатору

В задаче shadowJar нужно добавить к jar-файлу свою фамилию.

На рисунке 15 представлен измененный код задачи shadowJar, а также сгенерированный файл с расширением jar с фамилией.

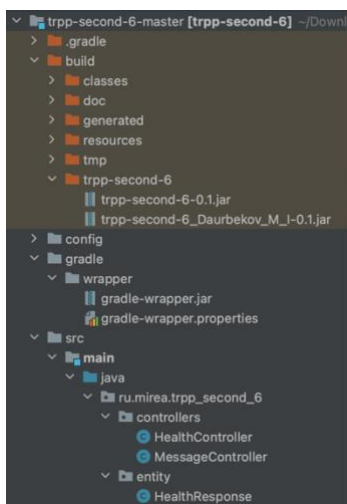


Рисунок 15 – измененный код задачи shadowJar

Рисунок 3 – сгенерированный файл с расширением jar и фамилией студента

Для последнего задания необходимо выполнить задачу checkstyleMain, посмотреть сгенерированный отчет и устранить ошибки оформления кода. На рисунке 16 представлено выполнение задачи checkstyleMain. На рисунке 17 показано исправление одной из ошибок, связанной с некорректным названием основного пакета. на рисунке 17 показаны ошибки оформления кода, а на рисунке 18 - исправление одной из ошибок, связанной с слишком большой длиной строки в файле. На рисунке 19 представлен отчет, сгенерированный после устранения ошибок.

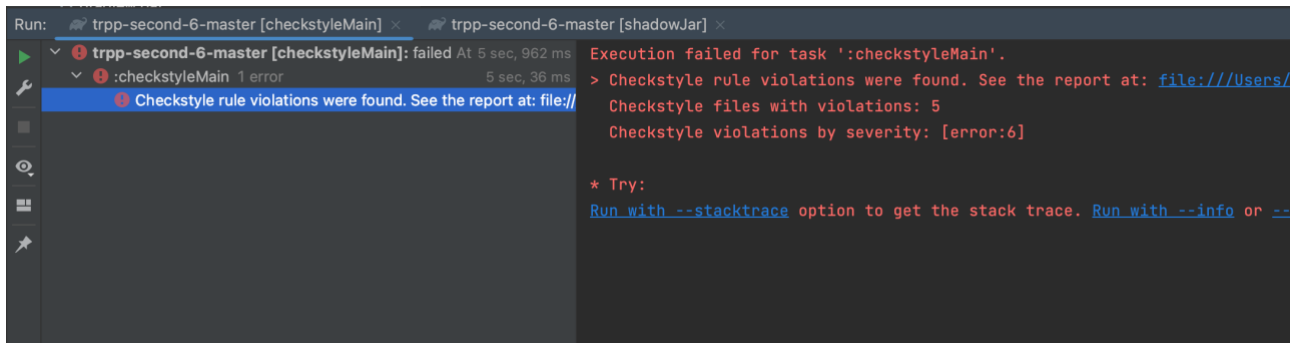


Рисунок 16 – выполнение задачи checkstyleMain

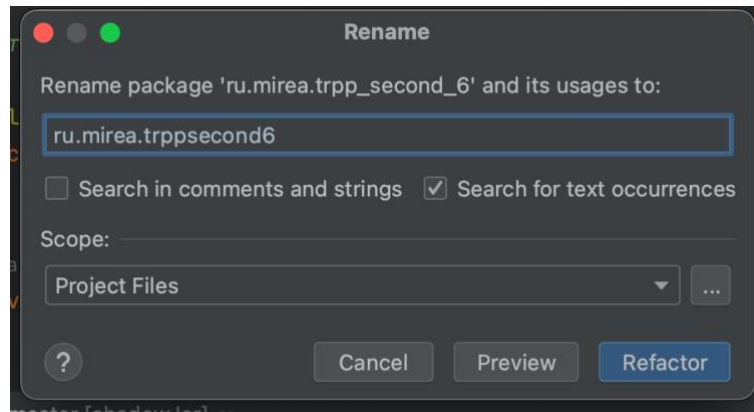


Рисунок 17 – исправление ошибки, связанной с некорректным названием основного пакета

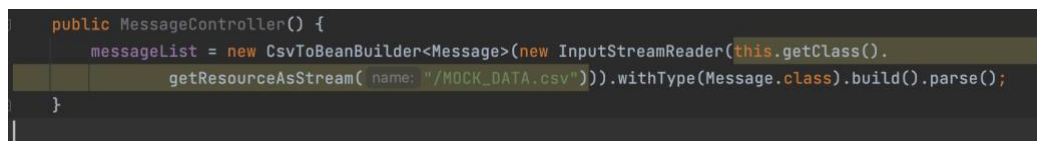


Рисунок 18 – исправление ошибки, связанной со слишком большой длинной строки в файле

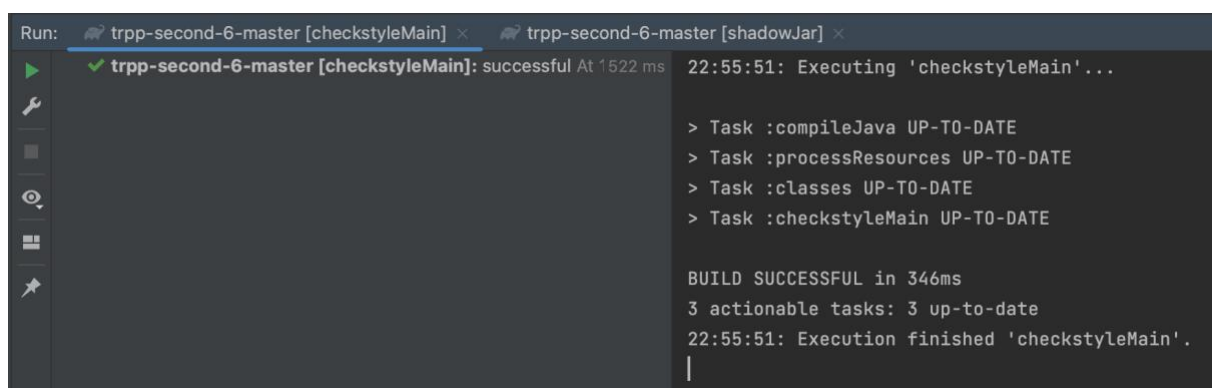


Рисунок 19 – отчет, сгенерированный после устранения ошибок

1. 2 Ответы на вопросы

1. Чем компиляция отличается от сборки?

Под компиляцией понимается получение объектных модулей из исходных текстов программных модулей. Под сборкой понимается получение выполняемого файла из объектных модулей.

2. Что такое система сборки?

Система сборки — это программное обеспечение, обеспечивающее автоматизацию сборки проекта. Основное отличие от IDE в том, что конфигурационный файл для системы сборки вы описываете в текстовом виде. 3. Что такое Gradle?

Gradle — система автоматической сборки, построенная на принципах Apache Ant и Apache Maven.

4. Что делает команда `compileJava`?

Данная команда компилирует исходные коды программы.

5. Что такое Postman?

Postman — это HTTP-клиент для тестирования API. HTTP-клиенты тестируют отправку запросов с клиента на сервер и получение ответа от сервера.

6. Что такое аннотация в Java?

Аннотации — это форма метаданных. Они предоставляют информацию о программе, при том сами частью программы не являются.

ВЫВОД ПО ПРАКТИЧЕСКОЙ №2

В ходе выполнения практической работы был изучен функционал и возможности системы сборки Grandle, а также были получены навыки по управлениями зависимостями. Еще был изучен программный интерфейс и был применен инструмент Postman для отправки запросов на сервер и просмотр ответов от него. Все поставленные задачи были выполнены.