**REVISION HISTORY**

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 31.12.2023 | 1.0 | Static Code Analysis results are added. | Azra Acil  Ahmet Pehlivanoglu |
| 31.12.2023 | 1.1 | Dependency Analysis results are calculated and added. | Mehmet Kutlutan |
| 01.01.2023 | 1.2 | Test Coverage results are completed and added to the Software Analysis Report. | Kerem Bekmez, Sevval Kurt |

**TABLE OF CONTENTS**

[**1 Introduction 3**](#_heading=)

[1.1 Document overview 3](#_heading=h.1fob9te)

[**2 Static Code Analysis 4**](#_heading=h.2et92p0)

[2.1 Tools 4](#_heading=)

[2.2 Results and Discussion 4](#_heading=)

[**3 Dependency Analysis 5**](#_heading=h.1t3h5sf)

[3.1 Tools 5](#_heading=)

[3.2 Results and Discussion 5](#_heading=)

[**4 Test Coverage Analysis 7**](#_heading=)

[4.1 Tools 7](#_heading=h.lnxbz9)

[4.2 Results and Discussion 7](#_heading=)

# **Introduction**

## ***Document overview***

This document presents and interprets the code analysis results regarding the VRS software development project. Code is analyzed by 3 different tools: 1) Static Code Analysis tool 2) Dependency Analysis tool and 3) Test Coverage tool. The first tool is used to reveal potential bugs that might be overseen during the testing process. The second tool is employed for evaluating the design quality based on the amount of coupling among the software modules and to what extent the code reflects the originally envisioned design. The last tool is used for measuring the coverage of unit tests in the project. Each section below is dedicated to each of these 3 analyses.

# **Static Code Analysis**

## ***Tools***

Checkstyle is a static code analysis tool for Java source code. Checks code against established code standards and style guides. This tool supports multiple code standards in a configurable manner and is also available as a plugin integrated with IntelliJ IDEA.

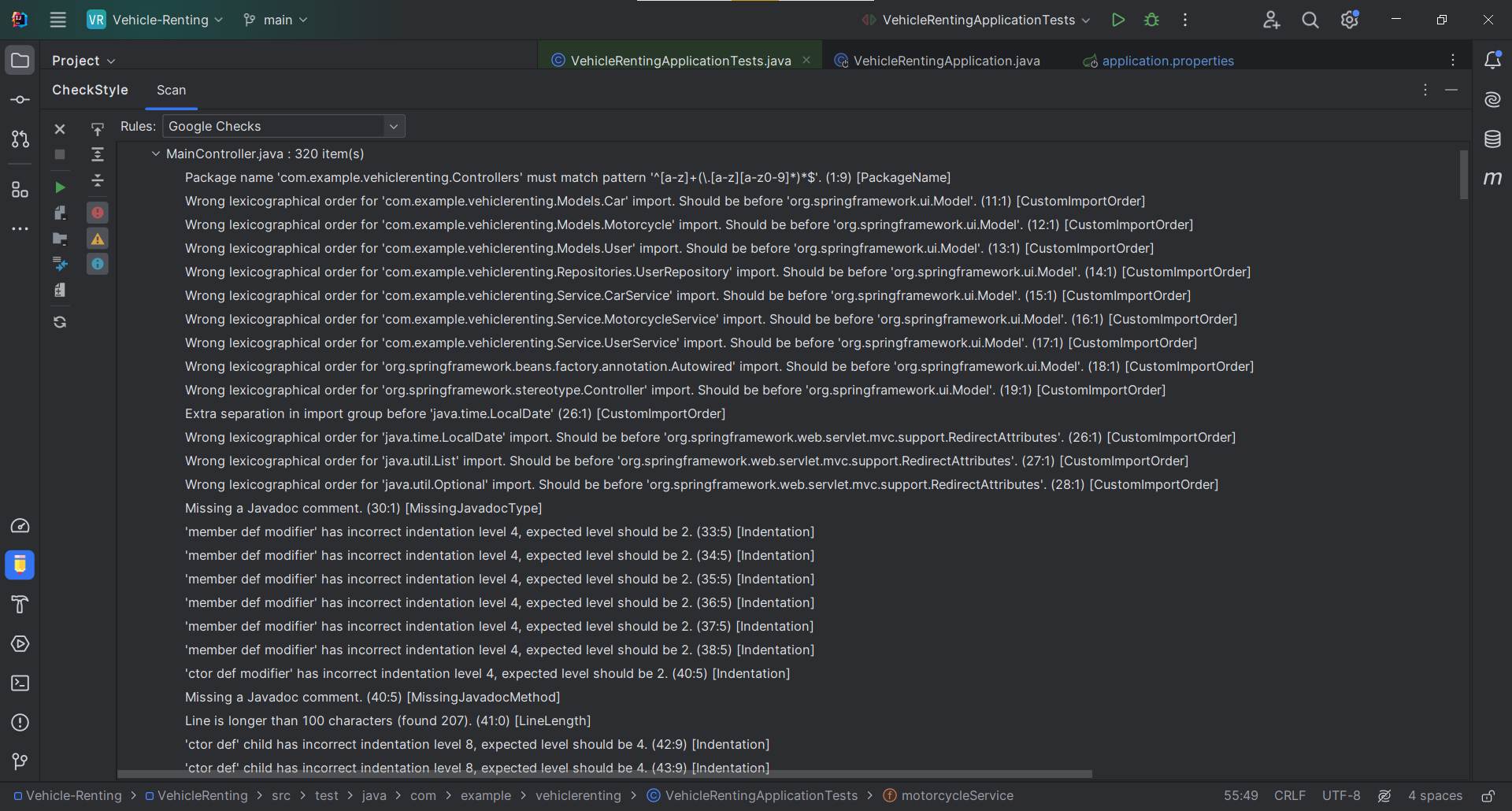
* CheckStyle, <https://checkstyle.sourceforge.io/>

## ***Results and Discussion***

For CheckStyle analysis, we will consider the MainController.java file as an example, which is our biggest, most complex file. The CheckStyle analysis of the MainController.java file returned 320 suggestions in total. None of these were determined to be critical; however, inconsistencies in indentation, line length, import order, and missing Javadoc comments were noted. Such style and format issues can significantly affect the readability and maintainability of the code, potentially affecting team collaboration.

We used a CheckStyle plugin, which provided us with this information in the IDE.

The picture below shows only part of MainController.java's suggestions.



Analysis of alerts that are reported by the tool:

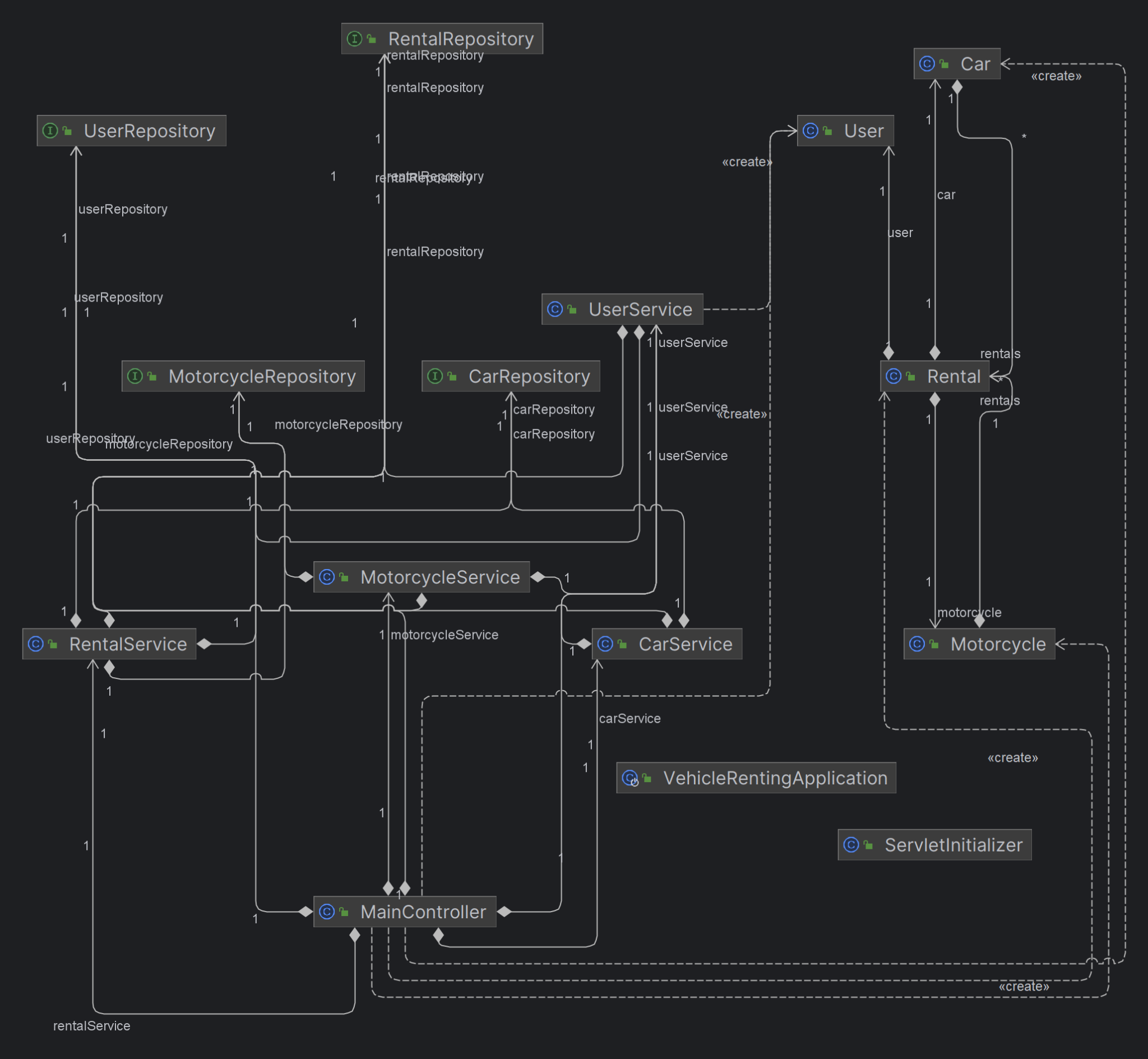
| **Type** | **Count of Instances** | **Severity** | **Count of Real Bugs** |
| --- | --- | --- | --- |
| Critical Error | 0 | None | 0 |
| Warnings of unused variable | 0 | None | 0 |
| Styling suggestions | 320 | Low | 320 |

# **Dependency Analysis**

## ***Tools***

We used the IntelliJ Ultimates Show Dependencies Diagram tool.

## ***Results and Discussion***



# of Edges = 22

# of Nodes = 15

Edge-to-node Ratio = 24 / 16 = 1.46

Tree Impurity = 2 \* (24 – 16 + 1) / (16 – 1) \* (16 – 2) = 8 / 91 = 0.08

It can be seen that the MainController is densely coupled. Our first design approach was to create different controllers for different functionalities, just like our models and services, but this caused problems with page redirects since we were using HTTP sessions, we could not fix this with using more than 1 controller. Hence, we had to combine everything under one controller. However all of our models and services are reconfigured whilst developing and as a result we have low coupling on our different models & services.

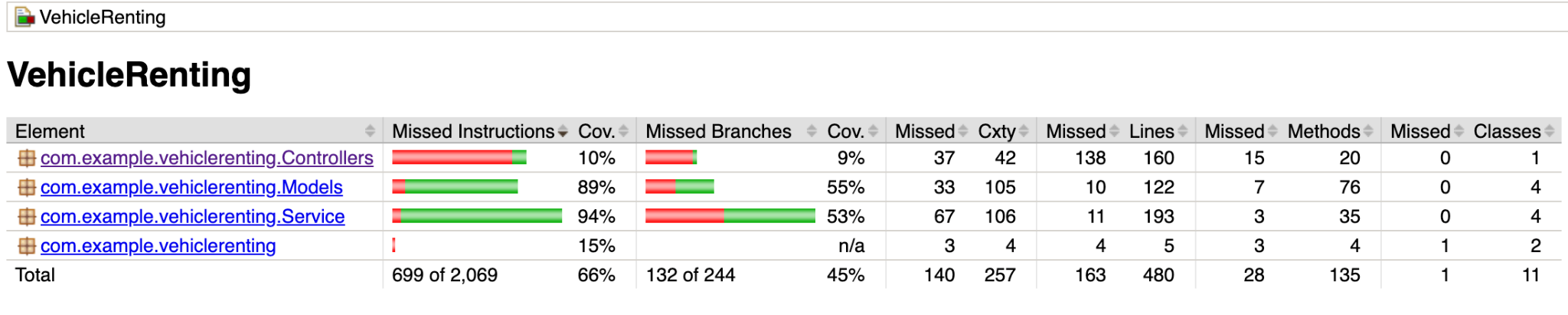
# **Test Coverage Analysis**

## ***Tools***

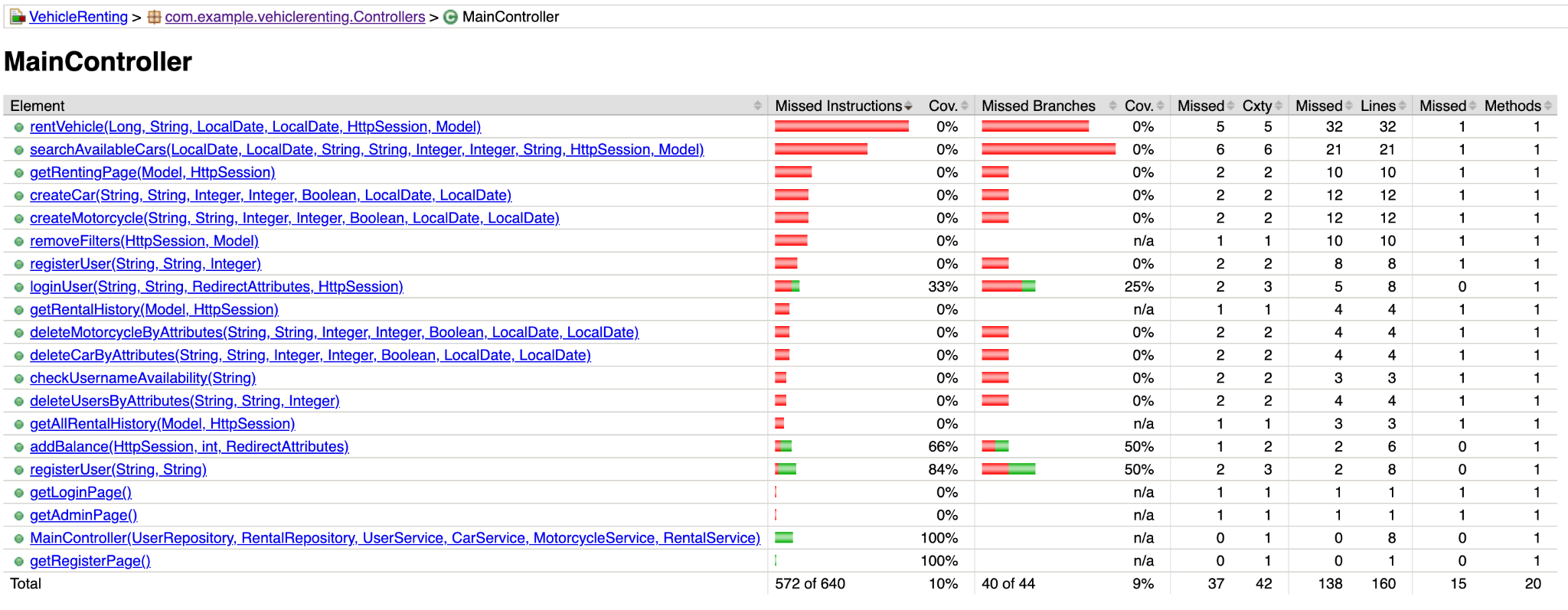
* We have built this project using Spring Beans, and to test the coverage of our code, we have selected JaCoCo as our Code Coverage Tool.

## ***Results and Discussion***

Below are the overall results of Vehicle Renting System Coverage. The testable classes for VRS were the Controllers, Models and Services.



Below we can see the poorly covered functions in the Controllers Package.



We have achieved high coverage for Models and Services, more than 85%. However, Because of the problem we encountered during the controller refactoring, coupling was high on our MainController. This resulted in poor coverage in our Controller package and MainController class as shown above. Although MainController coverage is low, assuming correctness in HTTP functionalities, and given the robust testing of Models and Services, we anticipate minimal issues.