Alexandru SAVA
VVS laboratory
2021 - 2022

Static and dynamic analysis

1. Static Analysis

The main static analyzer for ruby on rails code is rubocop. This gem (library) can also auto-correct most of the small issues it founds. But there are many rules that are not Correctable by themselves. The programmer has to look into the issue and try to solve it.

To run this static analyzer, I had to run the command: > rubocop

This will analyze your entire Rails project and show you the Offenses.

I won't display the entire log here, because with Rails you can generate a lot of code, but most of it is not as the style recommends it. So most of the offenses were like: "*Prefer single-quoted strings when you don't need string interpolation or special symbols*".

The entire logs can be found in the root directory of the project under 3 different files, captured in different moments of solving the offenses: staticAnalysisBefore.md, staticAnalysisDuring.md and staticAnalysisAfter.md.

In the first file you can find the log for running > rubocop. The analyzer detected 594 offenses.

In the second file you can find the log after running > rubocop -A. This command auto-corrects what is Correctable. The analyzer corrected 562 offenses.

In the third file you can find the log after running > rubocop. In this stage, I already made some programmatic changes in the code, aswell as defining some personal rules for the static analyzer (see detailed rules in the third file). The analyzer found 0 offenses.

As far as I inspected, most of the issues were tied to not describing certain classes, for better documentation, as well as many methods having too many lines - more than recommended for clean code, or lines having to many characters.

CyclomaticComplexity and PerceivedComplexity were some warnings in a part of generated code.

Style/Documentation was triggered mostly in parts of the code that was automatically generated or pretty self explanatory.

I also had a warning about defining a constant in a block for rspec unit tests was not recommended.

2. Dynamic Analysis

For the dynamic code analysis I used brakeman. Brakeman is a security scanner for Ruby on Rails applications.

Unlike many web security scanners, Brakeman looks at the source code of your application. This means you do not need to set up your whole application stack to use it.

Once Brakeman scans the application code, it produces a report of all security issues it has found.

Luckily no problems were found in my project, since is not yet very complex.

Brakeman performs many different checks in the code.

More details and the logs of the performed dynamic analysis can be found in the root directory of my project in the file: dynamicAnalysis.md.