The first step for any engineering role is a simple take-home programming exercise. We typically receive hundreds of coding solutions every quarter, and the first level of screening is automated. In this post, we aim to highlight some of the differences between solutions that are selected or rejected during an automated review.

Our automated review does a basic hygiene check when candidates submit solutions. It goes for a manual review ***if, and only if,*** it passes all the following checks. Even if one of these checks fail, the solution is rejected.

- Check if git history is present
- Check if tests and README file are present
- Check if the mandatory executable files(s) are present
- Run the code and check if the output is correct

Below, we will dive into a bit of detail for each of the above points. But before that, let me give a quick overview of our automated environment.

Our automated environment

Our automated solution checker runs on a Unix based machine. Hence, one of the mandatory requirements for the assignment is that the solution should build + run on Linux. We understand not everyone has access to a Unix based machine and there are a lot of Windows users out there. In such a case, we recommend creating a Linux based Docker container or a Linux based VM and work on it.

Some consider UNIX to be the second most important invention to come out of AT&T Bell Labs after the transistor.

— Dennis Ritchie

Candidates can also assume that the mandatory packages like java, Gradle, maven, ruby, bundle, go, Clojure, lein, pip, etc. will be available in the system PATH of the machine on which the automated solution runs. So one doesn't need to install these or set something like JAVA_HOME in the script.

Git History

A mandatory requirement for the technical assignment is the git history of the solution. We want to see how the code has evolved. This gives us an idea of how a candidate approaches the problem, rather than just solving the problem. This also helps us weed out solutions that are a mere copy-paste from the internet.

When we ask for a git history, we expect a candidate to create a local git repo, make as many commits as necessary, zip the local repo and send it to us; without ever actually running the command git push.

One of our mandatory requirements is that the solution should not be published publicly on the internet. So, when we say a *git repo,* we mean a local git repo and not a github (or gitlab, bitbucket, etc.) repo.

Tests

We give a lot of importance to automated unit testing. It's a good practice since it ensures any new change is not breaking the existing functionality. In the absence of tests, the submission is bound to get rejected.

It's not just the presence of tests. We also look for code coverage. If tests are present, but code coverage is poor, the submission will be rejected.

We've also seen submissions in which tests are "*not so good*". Diving into the details is beyond the scope of this post. But it might be helpful to delve into this article by Uncle Bob or this one by Martin Fowler.

README File

Needless to say, a nice README file goes a long way in onboarding new developers onto the project. So there should be a README, with all the relevant details that you may see fit.

Mandatory Executable File

Another absolute must is to have Unix executable file(s) as mentioned in the problem statement. The exact filenames and paths can be found in the problem description. The automated setup runs the files (with the exact same name and path) and checks if the solution is producing the correct output. In the absence of these files, our automated setup fails to run the solution.

Some submissions have the required executable file(s) but the automated setup still fails. This is mostly because there are a few manual steps that need to be executed before running the file(s). For example, some of the common errors we see:

- Unable to access jarfile target/project-name-1.0-SNAPSHOT.jar
- Could not find or load main class com.gojek.solution.Main
- Failed tests: testMainWithFile (com.gojek.SomeTest): /Users/abc/Desktop/input.txt (No such file or directory)

If we look at the above errors, a clear pattern emerges.

- A simple build command, for example, mvn clean build or ./gradlew clean build is not included in the executable file. In case of Ruby codes, bundle install might be missing!
- Candidates have either not compiled all the classes or have forgotten to include proper manifest in the jar file.
- The tests have hardcoded paths to files on the candidate's machines as a result of which tests fail on any other machine except the one on which they were written.

We have received submissions where there are clear step-by-step instructions on how to execute the solution in the README file. This is great and we really appreciate it. ❤ However, at the moment, our automated solution is unfortunately not intelligent enough to parse the README file and run the mentioned steps. (Though, we do aspire to build an intelligent system someday!)

Candidates have even submitted solutions with a language-specific executable file like .jar or .rb. But we are particularly looking for language-independent executable files.

Please note that this step also tests the candidate's end-to-end automation capabilities.

If all these checks pass, the solution goes for a manual review. An experienced software engineer looks into various aspects like code design, domain modeling, SOLID principles, naming conventions, appropriate usage of design patterns, etc. After an overall comprehensive review, candidates are called for the next round.
