FTR Platform Developer Coding Test

Aleksei Miros

Part 2:

1:

The other approach I considered when building the test application was a console application, so I will talk about how I would restructure my current program to a console application. The way the user would interact with a web application versus console application is quite different. My jumping off point would be extracting the major functions that aren't directly getting or displaying data. These would be the functions found in the utils directory. I'd then look to take the components created and turn them into a rough baseline for classes needed. I would also make a note of the restrictions and validations for user input that are currently in place. With the web application, a large amount of input sanitation is done by restricting possible inputs. More checks and error handling would likely be needed for a console application.

If I was going to stick with TypeScript, I would create a simple node console application. I could have classes for the console application housing the main functionality, a User, housing their input frequencies and associated logic, a Timer class, and a Fibonacci Sequence class used to generate the sequence and house associated logic. The UI would be far simpler, with input read in and feedback given via the console like the example given for this assessment. A possible challenge would be the need for the program to output to the console, while waiting for an input. A solution to this would likely be asynchronously getting user input for the main loop, while synchronously getting input for the setup information at the start (the delay).

2:

If I was going to be sending my application to production, I'd take a few steps. I'm assuming the application would grow in complexity and usage over time. The first would be to quickly go over the application and make sure there was no sensitive information set outside of environment variable.

Assuming the product would be worked on by multiple developers down the line, I'd set up the basics of version control, such as a repository with a develop, release candidate, and main branch, ensuring merging to a major branch requires a pull-request. A testing environment could also be spun up, with a second, user acceptance testing environment possible if the project and complexity grow.

I'd create a build pipeline with a tool such as Jenkins. Some key requirements I'd add to the pipeline would be to run unit and end-to-end tests, check for styling with a linter, check for security vulnerabilities, building the artifact and storing it on an private repository such as Artifactory, and automating the deploy to an environment (depending on branch type and user acceptance).

A simple hosting service would suffice to start with, moving to a cloud provider if usage increases. This would allow for additional security (if it was a managed instance) and higher up time. Multiple instances and availability zones with a load balancer could be used to ensure no downtime, and no-outage production deployments.

3:

I really liked the flexibility of being given a set of requirements and then being able to implement them however I wanted. Not having a strict time to complete was also very nice, allowing me to complete the challenge on the weekend. No single requirement was overly difficult, but combined provided an interesting challenge.

I also like the second part of the challenge. Being a developer is more than writing code so it's good to see assessment checking how someone thinks and communicates. A possible addition to part two could be a brief self-critique of the final product. This could provide insight into applicants that maybe had an idea of how to implement a solution but got stuck or ran out of time.This could also allow applications to discuss possible shortcuts or inefficiencies they implemented due to the nature of the application being an assessment and allow them to explain how they would have achieved the result in a working environment.