GiveDirectly Skills Interview for Colin MacAllister

Written Question

I've programmed professional code in a variety of languages, including Java, PHP, ColdFusion, ASP, Python, and more. But the one constant in my career has been JavaScript. Early in my career I used to lambast it as much as anyone, rebelling at the underwhelming attempt to graft object-oriented programming methods on top of what is at root a functional language. (I still believe that aspect of it makes it one of the *worst* languages when used that way.) However, over time people began to realize that the language underneath was quite powerful and elegant. Books like Crockford's "JavaScript: The Good Parts" cemented it in serious developers' minds as a viable language for any kind of development – not just tiny embedded UI enhancements.

However, I still missed the power of strongly typed languages for organizing large projects. Indeed, so many developers also missed this that Microsoft's Typescript was born. Typescript allows you specify in as strict a fashion as you wish classes and especially interfaces. This makes complicated projects much easier to handle as it self-documents the meaning behind every object/variable you've created. So, when you encounter a class within a method signature is some far-flung bit of your code, your IDE can tell you what the class was and give you an idea of how it was meant to be used. For small projects such as the one in the Practicum, JavaScript is fast and friendly. But for larger projects I turn to Typescript. (You can set how strictly type-safety should be adhered to, which I vary according to the size of the project and the turnaround time.)

If I had to list another language I dislike, in addition to incorrectly wielded JavaScript, I would put Java on that list. I used to love it, but they have tried so hard to stay up with modern functional programming that there are a number of features that are downright inscrutable. (And, I should know, because I taught a class on them last fall. The students were understandably mystified when I taught them about lambdas.)

Practicum

I have met all the minimum requirements. My work was made easy by using Express for mounting a server and Sequelize for object relational mapping -- Node with Express and Sequelize is a really easy way to mount RESTful APIs. Also, note that I defaulted to returning a couple of timestamp values that Sequelize provides free, rather than providing my own timestamp.

https://github.com/dcoli/gdapi

If I had more time I would create actual files to organize and contain the code for the Book and User models. In this simple example it's not a crime to do it the way I did it -- because the API is so simple, it's not terrible to include the data access framework in index.js. But for production code I’d like to have that Sequelize goodness squirreled away into its own model classes for Book and User. I would also use a formal router to handle the routes – but again, if the project were to stay this simple, there is no need.