Toy Development Project

The next step of the interview process is to complete a small “toy” development project. This project is not intended to take more than about 2-4 hours. Please stop once you get to 4 hours on this project. The purpose of this project is to:

- Give you a chance to get your feet wet with Golang if you’ve never used it before
- Show your skills at developing a web API endpoint
- Demonstrate your ability to create production-worthy code
- Show off your web UI skills

In order to keep the project as simple as possible, I’m providing you with a lot of the code you need to complete the project. That doesn’t mean that you need to use that code in your solution. You can modify the code to make it more testable, but hopefully the code I’ve provided will get you most of the way to a working solution.

The project

- Develop an API server in Go which will take a number as input and return the Fibonacci sequence up to that many digits. In other words, if I put in an input of 13, I would expect to get back 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144.
- For the purpose of this assignment, I am considering 0 to be part of the Fibonacci sequence. See https://en.wikipedia.org/wiki/Fibonacci_number for more information about the Fibonacci sequence and https://www.miniwebtool.com/list-of-fibonacci-numbers/?number=100 to see the first 100 numbers in the sequence.
- Develop a simple React web UI which will call the API and display the results.

Getting started

- For the React UI, we recommend using create-react-app https://github.com/facebook/create-react-app
    - `npx create-react-app my-app`
    - `nd my-app`
- In order to allow the React dev server running at http://localhost:3000 to proxy API requests to the Go API server, add the following to your package.json

```
"proxy": "http://localhost:8080",
```

- Start your React front-end development server by typing
    - `npm start`
- For the Go API, create a folder and copy and paste the code below into a file named main.go for your starting point

```go
package main

import (
    "fmt"
    "github.com/julienschmidt/httprouter"
    "net/http"
    "log"
)

func Index(w http.ResponseWriter, r *http.Request, _ httprouter.Params) {
    fmt.Fprint(w, "Welcome!\n")
}

func Hello(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
    fmt.Fprintf(w, "hello, %s!\n", ps.ByName("name"))
}

func main() {
    router := httprouter.New()
    router.GET("/api", Index)
    router.GET("/api/hello/:name", Hello)

    log.Fatal(http.ListenAndServe(":8080", router))
}
```

- Once you created main.go into a folder, type the following commands
  - go get -u ./…
  - go run main.go
- You should now have a running API server at http://localhost:8080
  - If you navigate to http://localhost:8080/api, you should see "Welcome!" in the browser window
  - If you navigate to http://localhost:8080/api/hello/<name>, you should see "hello, <name>" in the browser window
- Because we have our front-end server proxying request to the API server, you can also navigate to http://localhost:3000/api or http://localhost:3000/api/hello/there and see "Welcome!" or "Hello there!" respectively

What does success look like?

- I should be able to run your Go code and navigate to your API server at localhost:8080/api/fibonacci/2 and get the first 2 digits in the Fibonacci sequence. If I go to localhost:8080/api/fibonacci/24, I should get the first 24 digits, etc.
- I should be able to launch the web UI by running "`npm start`". I should then be able to type in my number of digits, hit a submit (or similar) button and see the results from the API service

Project modifications for your focus area

- If your focus is primarily on back-end development, feel free to focus primarily on the API server and only add the UI at the end if there is time. I'll expect you to have very high code coverage and cover all edge case. Bonus points for deploying the API server somewhere on the public web
- If you're most interested in front-end development, feel free to focus primarily on the web UI. You are welcome to write a simple Fibonacci generator in Javascript to get the results needed. Your UI should be elegant, easy to use and should be nicely designed. You should use good React design principles and should have tests for the UI.

Why is the unit test code important?

We write code every day that must be HIPAA compliant and which must work or it can cost us and many other companies lots of money. And, it can ruin the lives of people counting on us to do our jobs right. Because of that, we rely heavily on unit tests to ensure we're writing good code that is ready for production. I expect your code to be well tested. Well tested means that most if not all lines of code are covered by tests.

Are there specific tests we're looking for?

- In the case of a Fibonacci number sequence, you will reach a point where the Javascript Number or Golang int will overflow. Handle the overflow and return a valid error message
- Ensure valid input. For example -1 or "a" are not valid number of digits for the Fibonacci sequence. Give useful information to users on invalid inputs.

Project Submission:
Please send a link to the GitHub repository containing your code

What resources can I use?

- **Fibonacci Sequence in Go:** The Golang.org examples show how to do a Fibonacci sequence using a closure. Feel free to use that or other code online. Feel free to also modify it for your needs. https://play.golang.org/p/NeGuDahW2yP
- **Go API Server example:** Take a look at https://github.com/julienschmidt/httprouter and the Usage section for a full working example of a very basic web API server
- **Returning JSON data:** There are tons of tutorials out there on how to return data as JSON. Here's a good one that shows how to convert various data types into JSON. https://gobyexample.com/json
- **Go testing:** Take a look at: https://www.golang-book.com/books/intro/12 for some examples and ideas of how to write tests.

I look forward to seeing your work!

Sincerely,
Rob Archibald
Chief Technology Officer, 6 Degrees Health
robarchibald@6degreeshealth.com