Getting hired for a Go job

A corpo-friendly talk by Nick White

For golang Bristol++

2023-07-19

https://github.com/nickjwhite/jobtalk-golangbristol2023

Where I'm coming from

I had a relatively unusual career trajectory.

Studied religion and anthropology at university.

Did a couple of years doing web development and system administration.

Then spent 10 years as the sole software developer for niche research projects in Classics and English.

I wanted a change

My work had been super interesting, I learned loads about interesting subjects from world leading researchers, and I had almost complete autonomy in how I designed and implemented systems.

But I wanted to work with other technical colleagues to improve my knowledge and skills faster.

And I wanted to get paid better.

The job search took ages

It took me about 6 months to find the right job, which I attribute to:

- My unconventional career.
- I didn't know what I didn't know.
- I was quite picky in what I wanted (Go, remote, senior, money).

But I learned lots along the way, doing a bunch of interviews and coding tests, a lot of which should be useful to others.

CV

A good CV is hard, and I'm not convinced I entirely nailed it, but some good things to think about.

Your current CV probably isn't as well presented as you think it is.

Include *key skills* and a concise *personal statement* at the top.

Only include relevant information - no address of employment or contact info for past managers.

If appropriate, change job titles to match what you're applying for.

Use a bit of colour and space to make it flow - you may not be applying for a design job, but show you have taste, and make it easy for the person reading it.

Bullet points are better than paragraphs - this will be skimmed by a busy person.

Remember there is a good chance it won't be a developer reading it, so don't feel ashamed to include all the buzzwords you need to.

Interview

There are lots of different styles of interview.

Often it starts with a screening interview of 5-15 minutes, with a recruiter who doesn't know a lot. Tell them whatever they want to hear (based on the job advert) and how your skills make you a good fit.

Then interviews can be broadly categorised into 3 types (which may be combined together):

- Culture
- Technical
- Coding tests

Culture interviews

These can and should be prepared for.

Read the company website and job pages to get a sense of the *corporate values*. Do they see themselves as cool and progressive? Fun and relaxed? Serious and fair? Mirror that back to them; they're your values for the duration of the interview.

Expect and prepare for tricky questions like "tell me about an example of a time you showed leadership in a difficult situation".

Amazon are the kings for this sort of question, with their "leadership principles". They are also surprisingly great at providing preparation resources for them, which are super useful for any employer:

https://www.amazon.jobs/en/software-development-interview-prep

The key is to go through each "leadership principle" and list several examples in your career which demonstrate it. If you have time, break things down using STAR (Situation, Task, Action, Result).

Technical interviews

Read the job description closely, and think about what they're trying to build.

Think about how you would design this system. Don't worry if you haven't worked on something like that before, and it may be a bad design, talking through potential downsides or flaws in your ideas is great.

How does your current experience apply to this? Emphasise whatever parts of your experience make sense to make a coherant narrative that ends with you being hired.

Watch a few "systems design" example interviews on youtube, particularly this one: https://youtu.be/q0KGYwNbf-0

Coding tests

Some of these are live and timed, and some they send off to you and you complete and send back.

Coding tests: live and timed.

Practice working up little projects with no IDE. They won't care about perfect syntax, but you should be able to write something close to right from scratch.

These are generally somewhat abstract.

Sign up to https://leetcode.com and practise a bunch of their challenges.

Consider watching some beginner MIT computer science lectures on youtube to fill in gaps in algorithm knowledge: https://youtu.be/ZA-tUyM_y7s.

Coding tests: practical to complete in your own time

These can be surprisingly time-consuming, and can feel like you're doing unpaid work.

Think about splitting things up nicely to be extended beyond the scope of your test.

Write tests.

Write a dockerfile / makefile.

Write up some documentation, justifying why you've designed things that way, and things that should be improved in the future.

Bonus recommendation

https://cord.co was a very good site for finding jobs. You'll get a 5 minute interview with a recruiter quickly, and then be passed straight through to a real interview.

Fin

Q&A

–

https://github.com/nickjwhite/jobtalk-golangbristol2023