**Interviewing and applying TL;DR:**

They are more interested in your attitude (than qualifications etc) and what you can actually do.

**Tech tests**

**What is a tech test?**

When you apply to a company, you will generally have two kinds of interviews:

* a normal professional interview
* a technical interview

Sometimes they're in the same interview, sometimes you have more than one of either of these and sometimes you just have a technical test that isn't in normal interview format.

A tech test is any kind of test to determine your rough technical ability. You get a few kinds:

* a take-home build; a project build like the larger exercises we have done throughout the course
* whiteboarding tests; abstract questions that ask you to solve a problem (with or without a whiteboard)
* coding katas; problem-solving exercises
* pair-working; often work with one of their engineers to solve a small technical challenge (e.g. kata)
* technical quiz (e.g. how would you center a div in CSS)

**What to expect from each type**

**Problem-solving exercises**

This last kind are fairly easy to practice, and a good way of keeping your JavaScript sharp. They generally focus on taking and transforming data.

A few examples of this:

* FizzBuzz
* Reversing an array
* Building a function to sort an array

**FizzBuzz**

FizzBuzz is a challenge where you log the numbers 1 to 100. However, if a number is divisible by 3, you log Fizz instead of the number. If it is divisible by 5, you log Buzz instead.

If it is divisible by both (e.g. 15) you log FizzBuzz instead of the number.

This challenge is designed to test:

* loops
* conditionals
* some maths

And generally you will solve this by building a function, so it should test functions, too.

**Where to practice**

* Codewars
* Cyber Dojo (uses Test Driven Development)
* HackerRank
* Coding Dojo
* Exercism
* LeetCode

**Homework**

Set up an account on CodeWars, complete a kata.

Select at least JavaScript as your language.

Select the kyu range of exercises that you feel able to do. I recommend starting at 6kyu, and going up to 4-5 if you feel comfortable with that.

Pick an exercise. You will be given an exercise description, some example input/output, and a starting function to write inside. It will run tests against your function, until your function passes the exercise.

**take-home build**

A project build like the larger exercises we have done throughout the course.

You will receive:

* a list of requirements
* sometimes, a list of resources (e.g. an API link)
* sometimes, some existing/starting code

You will be given an amount of time to build this project.

These projects vary in size, from a couple of hours to build to days.

Your time is valuable, if you're unsure about the project feel free to ask me/each other/the recruiter for assurances such as:

* this code will not be used by them (or will be compensated if it is)
* if you are limited on time, that that won't be held against you

These exercises are often language agnostic (meaning you can use any coding language for them).

**Top tips**

Pseudo-code and document your project.

You do not need to finish every requirement, especially if it is time-limited. What you do complete should be working. It should be in a Github repo.

You should have a README that explains:

* how to open and run the project (or at least have it live) (e.g. npm i, npm start)
* which features you completed
* what technologies you use (and possibly why)
* what you would improve/change/add if you had more time.
  + e.g. if I had more time I would have added a filter functionality
  + e.g. if I had more time I would have refactored the project to use a CSS library

In your code, feel free to add explaining comments for yourself.

You can also leave pseudo-code comments for features you don't have time to build:

/\*

did not have time to add filter functionality, but it would:

- select the products

- loop through each one (using a filter)

- check whether the product had a sale === true prop

- update the state to show the current products with only the filtered items

\*/

Finally, mind your GitHub history.

* commit little and often
* use GOOD commit messages (e.g. "add changes" - bad. "Refactor filter function to use .filter not .map" - good)

**Ways to practice**

The best way is to practice on frontendmentor.io:

* put these builds on Github
* add them to your portfolio
* add any relevant info from the list above^ to your README
* as a bonus, either deploy them to Github Pages, or have a link to them using HTML preview (htmlpreview.github.io?url-to-your-html-file-in-github)

Other ways to practice:

Any of the projects or larger exercises we've done during the course.

**whiteboarding**

More common at larger companies. These days less common, certainly less common remotely.

Usually testing abstract thinking and problem solving. Sometimes assisted by someone else.

Sometimes these are algorithms... e.g. Big O notation. This is geared towards Computer Science grads. It is not ideal for bootcamp graduates who have focussed on practical application, over abstract theory.

You can look up ways to practice this, but if you focus on practicing problem solving using coding katas, you'll already be on your way to being good at whiteboarding problems.

Sometimes there is no right answer; you're not expected to always get it, sometimes the purpose is just to check your reasoning and problem solving process.

Try brilliant.org (Lucia cannot vouch for this) for practice?

**pair-working**

These "tests" are done with another developer and are meant to mimic how you will actually work at this company.

They can be a bit intimidating (live coding is scary!) but they're often one of the best types of technical interviews, and also the most flexible.

With pair coding you should:

* think out loud
* Google as you normally would
* and feel free to ask the other developer questions

The important thing to remember is that they don't expect you to know everything. E.g. if you remember that there is an inbuilt array method to loop over the array, but you don't remember whether it's forEach or map or you don't remember what order the parameters are... that's the type of thing you should say ("I know there's an inbuilt function for looping over the array, but I can't remember XYZ, I'm going to look it up") and then use the internet as you normally would to find the answer.

**Top Tips**

* Think out loud
* pseudo-code (this will help with nerves)
* ask for help as needed
* google as needed

**technical quiz**

"What are the pros and cons of using React?" "What CSS would you use to make a responsive set of columns?"

You can swot up (revise) on technical stuff. Practice coding, read through notes, etc.

Ultimately, the only way you can prepare for a company that does a tech text like this, is general preparation/ knowledge/practice - OR (ideally) ask someone else who recently interviewed there what questions they were asked.

**If you get questions you don't know the answer to?**

* Tell them you don't know
* If you can, offer to provide an educated guess
* regardless, ask them

"I don't know, but I could provide an educated guess. Based on XYZ, I think.... Is that correct?"

**Top tip**

If someone asks you What something is, it's often good to provide Why you use it.

**Tangents: Qs**

**Does everyone do tech tests, even at the senior level?**

Not always, but yes.

Google has a bunch of ways of hiring/talent-spotting developers; it will still put every applicant through tests.

**Should your Github only contain good/completed projects?**

No! However, you can pin repos of code you're particularly proud of to your profile.

The same goes for your portfolio: include incomplete projects.

Ideally, for ones that are incomplete, add some documentation explaining that it is unfinished and what you would do next.

**Should all the older (smaller) exercises be in one repo, or many?**

It's not hugely important... however, I'm erring on many. That way you don't have to worry about structuring them to be navigable (i.e. in a nice, tidy folder structure, and all documented).

However, for smaller/simpler exercises, you might want to put them on Codepen. You might want to put the link to this in your profile Readme, or Github profile in general, and in your portfolio site and CV if you choose to use it.

**Is it common that you will have a behavioural interview with e.g. HR before an interview w/ developer?**

Yes.

Generally the order of events is:

* tech test
* behavioural interview
* potentially an interview with someone more technical (e.g. a team lead)

**Building up your portfolio**

Use git! (and codepen)

Follow the advice for [practicing for take-home tests/builds]( Ways to practice)

**Non-technical interviews**

These are the "behavioural" interviews with someone like a manager or HR person.

**What they want**

They want to see "culture fit" (often).

Sometimes culture fit means: "someone who will enjoy working in our culture and will add value" (good)

Unfortunately, sometimes it is treated as: "Someone who is exactly like us" (bad)

Culture fit relates to:

* company values
* work/life balance
* responsibilities
* hierarchy/company structure
* social culture and behaviours
* and more

And there isn't a wrong answer to these. You can be NOT a fit for somewhere; that's okay. There is a place that will suit you and vice versa.

You can be honest about your values and look for the right job for you... That said, this is your first role. Prioritise getting a job. So long as you last at least 6 months (ideally a year), you can be much pickier for your next role.

Also, prioritise getting the offer. You can be choosy about it when you have more offers on the table.

**How to prepare\***

(\*applies to cover letters, too)

**Dress code**

Smart casual, usually. Dark jeans or trousers and a shirt or nice t-shirt and jumper, or a dress.

You could wear a suit to bigger companies (e.g. Deloitte, security companies, KPMG, some finance e.g. banks).

**Q & A Tangent**

**What job titles should you look for?**

Software developer/engineer

* likely to be a language agnostic posting
* Junior Frontend developer/engineer
* any JS framework, not just React
* including no framework (vanilla)
* you've been doing React for just 3 weeks; it's okay to switch to a different framework this early on JavaScript developer/engineer
* just be aware you might be asked to write Node; be upfront about existing experience
* this may be fullstackish (includes Node and a database)
* you can apply before you have taught yourself more than we have covered in this bootcamp Anything with HTML/CSS in the title
* however, this may involve working with other unfamiliar technologies
* may not use all the skills we have learned
* but would be fine Any of the above with Junior in the title or entry-level

Some others:

* QA testing
  + might not involve coding, might do (automated testing), but you migth have to teach yourself how to test
* Node engineer, TypeScript, etc (JavaScript family)
* Wordpress developer (you would have to learn PHP for this, but you could start on the frontend)
* working on eCommerce frontends
  + where you would focus on HTML/CSS (at first)
* more stuff in the coding family: data engineer (might need to learn Python), technical SEO + analytics

Base it more on the technologies and requirements in the job listing than the title itself.

**jQuery**

An older library that is still relatively widely used because it supports older browsers very well, and because it used to be the main/only(ish) library for JS developers, and because it's easy to learn.

A company that uses jQuery is more likely to use var than let or const, and the function keyword.

**What tech stacks should you apply for?**

Anything that uses JavaScript and/or HTML + CSS, or is language agnostic.

Sometimes you'll see roles that say "Experience in a language such as... XYZ". You can apply to roles that use other coding languages... but it might be easier to apply to ones that use the tech that we have learned.

**What if they ask for/mention technologies we haven't covered?**

e.g. tools and skills such as Continuous Improvement/Delivery pipelines (CI/CD) such as Travis e.g. other frameworks/languages in addition to the stuff we've learned (e.g. TypeScript, Angular, Less)

Apply. So long as it uses the skills we have learned, don't stop yourself from applying to a role where you have experience in several of the technologies listed.

**What if they ask for a certain amount of experience (e.g. 2 years)?**

In general, I recommend applying for roles that ask for up to 2 years experience.

A better indicator is salary. If the salary range is between 20-28k (or 24-30 ish in London), it's for you. If it's a bit higher, feel free to apply anyway.

**What if they ask for certification (e.g. degree)?**

Apply anyway! If everything else is mostly a match, then apply.

1. University grads have a lot of theory, you have a lot of practical.
2. Sometimes companies just put this in as an early filter, not because it matters.
3. Degrees don't help with building frontend code as much as backend code.

The companies that really actually care about this? You probably don't want to work for them anyway.

**Do you need to prove you finished this bootcamp?**

Not for every role, some companies will ask; you will receive a digital certification. However, if they ask use me/the university as a reference.

**What if it's a great company with no junior roles?**

Contact them and ask! "Cold-approach"; calling or emailing them (go by their recruitment email address on the website) to ask whether they would be open to interviewing you for a junior position.

**Is it enough to just showcase our portfolio since we don't really have commercial experience?**

Yes, it should be.

If you have transferable experience (e.g. working to deadlines or requirements or as part of a team), you can use this to answer interview questions.

**Best places to look for jobs?**

All the normal jobhunting websites (Indeed, Monster, etc).

GradCore provides a job board.

Additionally, look at the websites/Careers page of companies you like.