G2i

How to Hire a React Native Developer

A comprehensive guide to identifying and hiring the right React Native developer

About the Author

Andrei Calazans

React and React Native expert, Andrei has helped ship React Native apps to millions of users. For the past 3 years, he has interviewed, reviewed code, and worked with some of the vetted engineers of G2i. He is in love with the trade, and in his spare time you are likely to find him coding.

Table of Contents

What is a React Native Developer? 3–6

What Should React Native Developers Focus On? 7

How to Write a Better Job Description for React Native Developers 8–9

Responsibilities and Requirements 10–11

How to Approach Hiring a Junior-Level React Native Developer 12–13

How to Approach Hiring a Mid-Level React Native Developer 14–16

How to Approach Hiring a Senior-Level React Native Developer 17–19

How to Determine What Type of React Native Developer You Need 20–21

Questions to Ask When Interviewing a React Native Developer 22–29

The Lingering Problems with Technical Interviews 30

Finding A Universal Standard for Different React Native Developers 31–33

What is a React Native Developer?

Simply put—developers are makers. They build apps, services, and interfaces into existence. But, no two developers are exactly alike. They might have different skill sets. They might be specialists in different programming languages. Or, they might be experts in building one particular type of app. This principle applies to React Native developers as well. So, let's dive into what makes a React Native developer, what type of work they focus on, and the tools they use to get their work done.

React Native developers typically build native apps on the two dominant mobile platforms—iOS and Android. But, that universe is rapidly expanding. Now, React Native developers might build an app for Windows or Mac OSX, or construct an app that can run on a TV.

The primary focus of a React Native developer is building those apps across platforms, and across industries. React Native developers use flexible tools like JavaScript that make their work both accessible and valuable across various industries and at leading companies like Microsoft, Facebook, Twitter, Discord, Teslas, Bloomberg, Wix, Artsy, and others.

This should come as no surprise, but a large part of a developer's day is writing code that will eventually

grow to define and shape a user's interface. You may have guessed, but I'm a React Native developer myself. When I'm working on a new application, two critical parts of my journey are writing UI code and mapping data that's ferried into and out from the application itself. Although, there's always more work to be done.

Some roles require the React Native developer to build, test, and release the app. This means developers should have a good understanding of build workflows, continuous integration, and test tools.

We've covered a few tasks developers accomplish when building apps. Now, let's unpack the various tools and frameworks they use to power those apps.

React Native Tools and Frameworks

The React Native Framework consists of JavaScript code that is embedded inside a Native app and interpreted using a JavaScript engine. Depending on the platform, the framework logic is written either in C++ or with the platform's default language. This fact makes the React Native developers' job a bit harder since they are writing JavaScript code inside a Native App. This dynamic results in more exposure to the native software development environments.

React Native developers have a high-level view and deep understanding of how and when to call the native layers to leverage features that might be unavailable within the confines of the React Native framework. To do this, developers leverage tools like Xcode and Android Studio while dealing with third-party native dependencies.

It is not uncommon for React Native job listings to request developers to have extensive experience with native development. Some teams might even actually prefer developers with a native development background, rather than a web-based background.

That preference might be a little misguided given that a React Native is more exposed in the JavaScript layer.

Let's breakdown a React Native developer's role so we can see its component parts and understand what defines a developer's day-to-day work:

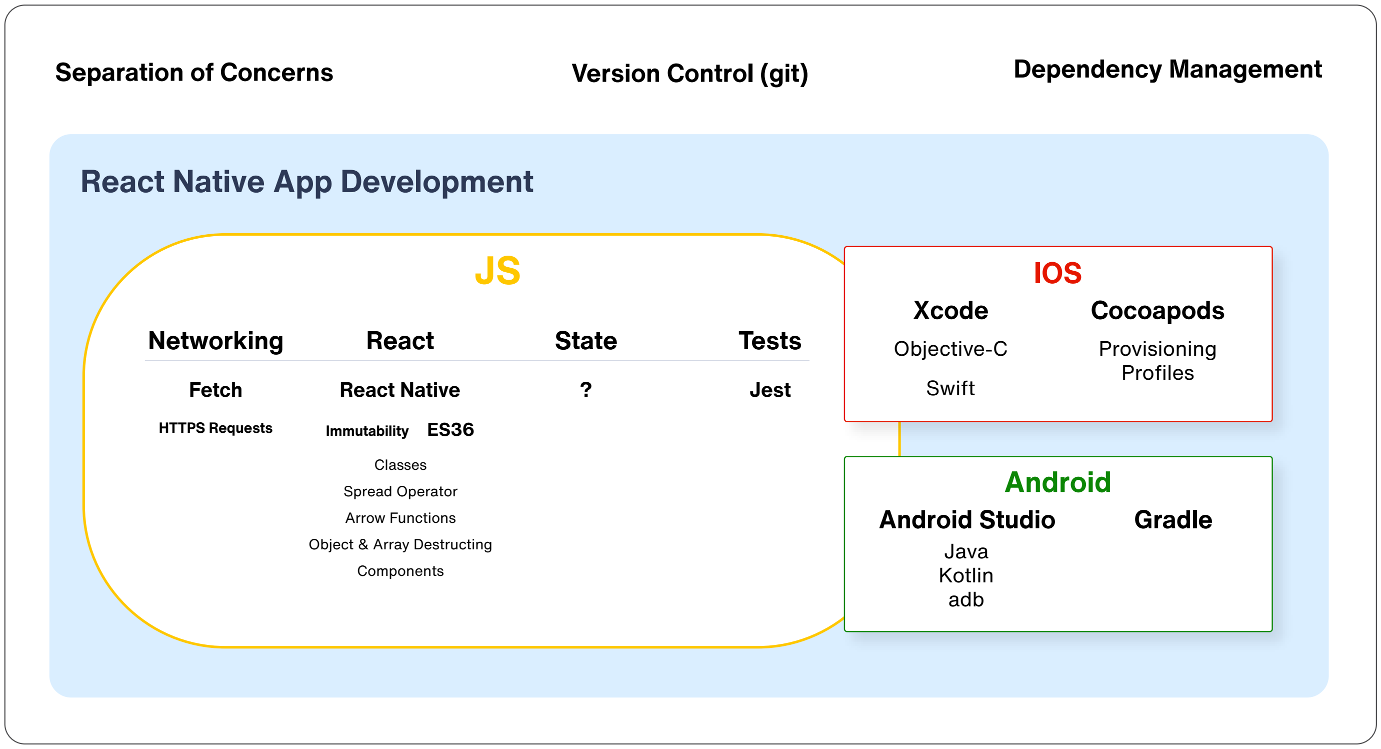
- ✓ Writing JavaScript
- ✓ Handling network data and local state in JavaScript
- ✓ Writing React components
- ✓ Writing React Native layouts
- ✓ Integrating native modules
- ✓ Building and releasing apps

A developer's first goal should be to support the full lifecycle of the product, taking what they've built, getting it through testing, and to the end-users.

While some teams can establish a certain paradigm to follow, like a functional parvadigm, codebases written with JavaScript are always flexible, willing to mold to whatever that specific company or set of developers consider the best of all the worlds. This is why it's so important for React Native developers to have a strong foundation in JavaScript.

React Native app development is composed of knowledge bubbles. A React Native developer is first a software developer before anything else. Not every developer will go deep into the platform-specific bubbles, and a shallow understanding of some of these bubbles is often enough to be productive.

Software Development



How the Pieces of the React Native Puzzle Fit Together

When someone is writing a React Native app they will use JavaScript and React to create components. While two ingredients might seem easy to manage, there are so many variables that a developer needs to wrangle to produce a functional application. Let's take one example — deciding between component types.

Before React 16.8 came out, a developer would either use a Class component or a functional component when writing a new component. The decision to go with a Class Component normally meant that the developer needed to use the lifecycle methods and or local state.

In order to use a Class component, the developer had to know how to instantiate an ES6 JavaScript class. They also had to have an understanding of the pitfalls of using this keyword within a method of the class and how class fields and arrow functions could fix this keyword problem, which React lifecycle methods were adequate for them to use, the impact of inlined values, how to use the local state API avoiding mutation and a whole lot more. This is not at all an exhaustive list, but it illustrates the depth granular knowledge a React Native developer has to have when writing a new application.

So, what should you look for when hiring a React Native developer?

What to Look for When Hiring a React Native Developer

While anyone can write React components and React Native apps. It takes someone with a certain level of fluency to excel at React Native development. Some companies don't have the luxury of waiting for a developer to ramp up on React Native after they're hired.

They might require someone to be productive from day one. And in these cases, coming in with a deep knowledge of React and React Native is a must.

Thus, a React Native developer is someone who is very confident with JavaScript, has a strong understanding of React's model, knows React Native's layout components, and might have either a shallow or deep understanding of the platform-specific development tools.

What Should React Native Developers Focus On?

Some issues React Native developers see every day. Others pop up every once in a while. So, what issues should developers focus on, and what issues might not deserve their full attention?

The exposed native environment means developers can easily reach out to the native layers too, allowing us to get more done with a few more lines of code. However, these are rare instances, and expecting a JavaScript developer to write high-quality Objective-C or Java code is a large ask.

React Native developers aren't experts in Objective-C or Java, those two domains are their own discipline. So, while a developer might be familiar with those two languages, it might take them more time to write that native code.

On a personal note, I believe that every React Native developer should be a software development generalist, amassing a solid, broad foundation before holding expertise in any single language. Feeling comfortable with reading and possibly writing code in different languages is incredibly important.

Common Mistakes to Avoid When Writing Job Descriptions

A mistake I see in many job listings when classifying the requirements of a React Native Developer is fluency in Objective-C and Java. Someone who has been writing React Native apps for the past four years is unlikely to have high-level fluency with native languages. Their job simply does not involve writing code daily in these native languages.

You might expect an electrician working on your house to have some familiarity with construction based on the environment they typically work in. But, that's not their specialty. You didn't call them for their expertise in that field. They have a specific task to accomplish and domain to focus on. The same principle applies to React Native developers.

A React Native developer is someone who uses modern JavaScript to write React components, composing UI alongside React Native components to shape a user interface. Part of this job is getting data from their network and mapping it back to the user interface.

Occasionally, developers are required to integrate a native module, usually provided by external parties, to access a feature or API unavailable in the React Native framework. These are the core proponents of a React Native developer's job.

How to Write a Better Job Description for React Native Developers

Typically, you'll find job descriptions presented in four parts: a job summary, responsibilities and duties, qualifications or requirements, and benefits.

Companies have to package this information, present it, and set expectations carefully and consistently. When it comes to packaging, we're not saying there's one service you have to post to or one font you have to use. Packaging information more refers to laying the critical job information out clearly and providing context that a developer can use to determine if the job is right for them. Whether that happens on a job board, in an email, a PDF, or even video is up to you.

At G2i, we sometimes don't write job descriptions — we film them. Using video services like Loom, you can make a personal impact and carry a candor and energy in your job description that might not come across as easily on LinkedIn or in an email inbox. In this environment, the prospective developer gets a better sense of your team culture, and your team member's personality, building a sense of connection before they even fill out a job application.

For example, if we were posting a job description for a Senior Developer at G2i, we'd use Loom to explain more about the role and give potential candidates a more personable, friendly deep dive into what the role entails. Here's one example of how we use Loom.

There are certain phrases or details that might not make the cut when you're writing a job description, yet are essential to the role. Instead of writing phrases that point at those critical details, you can call them out by name in a video. Imagine if we tried to pack a phrase like this into bullet points within a job description:

"The technical skillset required to build G2i's tools, features, and products when you can't just drop by someone's desk to ask them a question requires a level of experience that's a lot to ask for. So, we're looking for someone that learns fast and wants to see their development skills grow and thrive in a company that actively encourages on-the-job growth and learning."

Now, think about how you would have to condense all of that valuable information down to fit in a job description. You might end up just using bullet points that say "fast learner" and "remote-friendly team" or "self-starter" but none of those terms accurately describe the context of the job or the amazing opportunity. They're more geared around what your company needs, rather than what your company can provide to a candidate.

At G2i, we're value transparency, we value human connection, and champion a people-before-product style philosophy.

We care not only about the work we do but the happiness of our coworkers and colleagues at work. When you're hiring for a developer, it's important to keep those issues in mind. You're not just trying to find someone to build an app, you're trying to find someone you can build multiple apps or multiple services with. That requires a relationship that's founded on trust and is more than transactional. At G2i, we try to build those relationships starting from the job description. That's why video is such a powerful tool for us.

So, how do you craft a job description that accurately represents your company, your ideal candidate, and the market?

How to Approach Writing a Job Summary

From a developer perspective, I am curious to know what I will work with, what the industry is like, and what challenges I might face. What are the points that would attract a developer to your company? What is unique and required for the developer to know upfront?

Here are two templates you might find helpful when writing your job descriptions. The first is a Mad Libs-style template you can fill in yourself. The second is an example of a more fleshed out job description you might find when job hunting, or doing tape study on React Native developer job descriptions.

A Fill-in-the-Blank Template

We're looking for a React Native Developer to contribute to the development of our [industry] app. As a full-time contributor, you will work to [challenge 1], [challenge 2], and [challenge 3]. The ideal candidate is someone who [quality 1], [quality 2], and [quality 3].

A Filled-In Job Description Template

We're looking for a React Native Developer to contribute to the development of our marketplace app. As a full-time contributor, you will build a pleasant and fluid user experience; help establish best practices for the longevity of our Mobx and React codebase; and work together with a team of designers, developers, and stakeholders to further develop our product. The ideal candidate is someone who is comfortable with writing user interfaces with the React Native framework, has an interest in front-end architecture, and works well in a team environment.

Responsibilities and Requirements

Before we dive into responsibilities and requirements, let's make sure two things are clear.

1. There's a strong connection between responsibilities and requirements, even though you might list them in two separate sections on your job listing. If your role requires someone to architect the initial structure of your codebase, you have to make that explicit and clear in the requirements Don't expect the developer to assume that's required of them based on their skillset. Requirements and responsibilities give developers a good bead on what you expect of their skill set.

2. The hiring industry typically segments levels of experience by junior, mid-level, mid-to-senior, and senior.

For organization's sake, we'll follow the same naming convention here. But, we at G2i do not believe there's a 1:1 correlation between years of experience and the level of skill a developer has. Brilliant developers can code at a senior level with a few years of experience. Developers finding their niche might have several years of experience, but write code at a mid-level because they just switched their engineering focus. These distinctions should serve as a lesson to consider the developer in their entirety, not just the timeline of their resume. Depth-of-knowledge is more valuable than years of experience.

Education and Computer Science backgrounds should not be treated as requirements, but rather as a nice-to-have. It is nice for someone to have a high overview of the computer science field, but not a requirement. This misconception is slowly fading despite degree requirements still lingering in some job descriptions.

The folks who say that degrees are required to be considered might be relying on an arbitrary rule without considering what purpose the rule serves. If one applicant has a CS degree that didn't cover React Native, and another applicant has four years of experience in the job market building React Native apps, but no CS degree, who is more qualified for the job?

Designing front applications is a specific practice. It takes experience, not just a degree.

Experience, Expertise, and the Libraries In-between

Remember our electrician analogy from earlier? Let’s consider another analogy to better understand the state of building with React Native. As a refresher, JavaScript is the foundation of React Native development. React is the primary tool we use on top of that foundation. Everything else is replaceable for a React Native developer.

Alright, time for this butcher analogy.

A butcher might have a specialized knife to slice and dice meat. But what good is the knife if he doesn’t know anything about sourcing the right cut of meat? In this example, a state management library or a utility library like Lodash is a proverbial butcher’s knife. This knife is useless if the butcher does not know the types of meat and cuts.

Okay, now let’s imagine that this butcher is wildly well-versed in cuts of meat. He can cut a backstrap, carve up a sirloin, or get you a nice filet. This butcher would likely still be able to perform his job even if he was relegated to use a knife he only had a passing familiarity with.

The React ecosystem is rich in different options of libraries that are like that butcher's knife — they might not be perfect, but they’ll get the job done in capable hands.

There is no need to require a deep understanding of a specific third-party React library like Redux, Lodash, or RxJS. As long as the developer you’re hiring has a strong understanding of JavaScript and React, they will be just fine. Most third-party libraries can be learned rather fast.

In the current age of state managers, there are many options and each carries with it a set of patterns. A state manager is often composed of a core idea and a few APIs. When you list fluency of a specific state manager as a must-have, or requirement for the job, you might filter out people who can easily learn a state manager library in a few hours.

How to Approach Hiring a Junior-Level React Native Developer

When hiring for a Junior role, expect a bit of hand-holding. The best environment for a junior developer is to work with problems that have clearly defined boundaries and goals.

Not everything a developer is responsible for requires him or her to have previous experience. Part of a developer's job is taking his or her foundational knowledge and figuring out how to solve the task at hand.

The core of a developer's job is researching and learning. A junior developer can be expected to succeed if he is given challenges that are slightly beyond his capabilities but well defined. As a company that hires junior developers, this should be part of your managing strategy for your junior developers to grow professionally and generate more value for your company.

For a React Native role, you can expect a junior developer's day-to-day to be full of writing JavaScript code with the React Framework in order to either connect data coming from network requests or to write user interface components with React. There are also the soft-skills related to working with other individuals to prepare and understand tasks.

Typical Junior-Level Responsibilities

Writing JavaScript code following best practices.

- Constructing and organizing a component tree user interface using the React Framework while following best practices
- Writing React Native layouts and leverage the React Native open-source ecosystem
- Debugging React Native applications using debug tools.
- Integrating React Native Modules into the app
- Learning new JavaScript libraries and adapting to the new environment
- Working with a team to contribute to a codebase while keeping a healthy environment

Qualifications to Look for in Junior-Level React Native Developer Roles

For a junior role, the qualifications are heavily dependent on your company's work environment and if it is equipped to accelerate a junior's growth. Not every company is suited to receive developers that require a lot more learning.

There is a risk on investment if you don't take this into consideration since the return value from a junior role matches more of an exponential curve where initially it stays close to zero for the first few months until they can be considered productive. There are junior developers that are closer to mid-level due to either their learning capabilities or previous experiences. These developers can be productive from day one. Nevertheless, variables like how well organized and documented your codebase is, plus how healthy the work environment is all have an enormous impact on someone's onboarding productivity.

Junior-Level Qualifications

- Fluency in modern JavaScript (ES6+) and the ability to make decisions on the best approach to solving problems with JavaScript.
- Fluency with React.
- Capable of writing UI layouts with React Native components.
- Comprehension of version control and git.

It is expected and natural for a junior developer to be learning on the job. At G2i, we don't expect a junior developer to have previous experience with everything, so some qualifications can be considered bonuses or nice to have.

+ Understanding of Native Modules and how it can be integrated with each native platform.
+ Experience with codebases that leverage X state management solution.

Ideally, the idiosyncrasies of your codebase should be stated as bonuses since every codebase will diverge in either using or not using TypeScript, a state manager, or some other sort of pattern like Functional or Object-oriented programming. In any case, these are all tools a developer can learn while onboarding.

How to Approach Hiring a Mid-Level React Native Developer

At this level, a developer is more confident in their knowledge. There is less hand-holding here and you can expect a more concise contribution to the project as a whole: code contributions, task estimations, and technical decisions.

While you can still view any developer's role as a learning process, or a process of sharpening their skills, at this level the developer has a foundational skillset and can offer their opinion on tasks as opposed to simply accomplishing tasks presented to them.

As in any other trade, as you gain experience you also shape your opinion of how things should be done. From our experience at G2i, understanding a developer's point of view about coding can make or break a match between a company and a developer. When the company's outlook on development wildly differs from the developer's, it's hard to bridge that gap.

A mid-level React Native developer has a clear picture of the pros and cons of React Native itself. Working with React Native development for well over a year, you start to learn what works and what doesn't.

Beyond writing React Native apps, we expect mid-level developers to be more productive when writing these applications. This is why knowing the shortcomings of the tool is important. You can move faster when you know your instrument inside and out.

Typical Mid-Level Responsibilities

- Writing JavaScript code following best practices.
- Constructing and organizing a component tree user interface using the React Framework while following best practices.
- Writing React Native layouts and leveraging the React Native open-source ecosystem.
- Debugging React Native applications using debug tools.
- Integrating React Native Modules into the app.
- Learning new JavaScript libraries and adapting to the new environment.
- Sharing opinions on patterns to follow on a React Native codebase.
- Contributing positively to the Mobile UI experience of the app.
- Working with a team to contribute to a codebase while keeping a healthy environment
- Reviewing code from peers.
- Using git for version control following best practices.
- Contributing to the project planning phase by helping estimate and break down tasks.
- Raising critical paths during the project development and proposing solutions.

You can expect a mid-level developer to lead development if necessary. In some companies, they are even referred to as senior developers. The more they own a project, the more it is likely they'll deliver.

Most teams have a high number of mid-level developers. They are competent with the tool and even capable of mentoring junior developers. But, mentoring also depends only on their personal soft-skills, and their cultural fit with junior developers.

Qualifications to Look for in Mid-Level React Native Developer Roles

As we progress into the higher-level roles, developer knowledge expands into new areas.

- Fluency in modern JavaScript (ES6+). You can make decisions on the best approach to solving problems with JavaScript.
- Fluency with React.
- Knowledge of how to write UI layouts with React Native components.
- Understands version control and git.
- Experience with the Android and iOS development environment.
- Ability to estimate how long a task will take.
- Understanding of a Native Module and how it can be integrated with each native platform.
- Experience and understanding of codebases that leverage X state management solution.
- Ownership of the product you are developing and you are interested in the intricacies of the product you are developing.

How to Approach Hiring a Senior-Level React Native Developer

You don't need a senior developer to write React Native code. Folks in this level role are typically focused on higher-level work. You need a senior developer to architect a maintainable and scalable software codebase that can last. You need a senior developer to make the technical decisions that impact your company.

The more someone knows, and the higher their ability is to share their knowledge, the more likely they are to be seen as senior-level professionals. You don't want to hire someone simply because they've worked in the industry for more than ten years. You want to hire them for their expertise and competency. You want to hire someone who can lead a team to deliver a product. The goal of any senior developer is to have a macro overview of development, to learn from their mistakes, and apply the lessons in an actionable manner.

You shouldn't expect a senior-level developer to solely have React Native experience. React Native is young, and senior developers who are well-versed in the industry have likely worked with other languages, tools, frameworks, and APIs.

This can also be true for mid-level developers. From our experience interviewing at G2i, we have had positive experiences interviewing developers who migrated from different tech stacks. They often have a better understanding of the problems React and React Native are built to solve.

The speed in which a developer can learn a new framework also dictates the level of someone's experience. For a trainee, when they are learning React they are often also learning programming and mobile app development at the same time. This makes it harder for the developer to distinguish each layer abstraction, causing them to visualize it all as one. More experience with different stacks and programming overall allows the developer to pick up new frameworks and programming tools faster due to a better understanding of each abstraction-layer.

Typical Senior-Level Responsibilities

- Writing JavaScript code following best practices.
- Constructing and organizing a component tree user interface using the React Framework while following best practices.
- Writing React Native layouts and leveraging the React Native open-source ecosystem.
- Debugging React Native applications using debug tools.
- Integrating React Native Modules into the app.
- Determining what libraries and tools to use.
- Setting patterns and standards on how the codebase should evolve.
- Guiding the team to follow correct Mobile UI experiences.
- Influencing and establishing healthy company relationships.
- Mentoring and helping others increase their knowledge of the software and tools they are using.
- Reviewing code from peers.
- Use git for version control following best practices.
- Lead the project planning phase.
- Raise critical paths during the project development and propose solutions.
- Working with engineers, designers, product managers, and senior leadership to contribute to the final product.
- Optimize code for performance, stability, and maintainability.
- Help define application architecture.

Qualifications to Look for in Senior-Level React Native Developer Roles

- Fluency in modern JavaScript (ES6+). You can make decisions on the best approach to solving problems with JavaScript
- Fluency with React
- Understanding how to write UI layouts with React Native components
- Understanding version control and git
- Experience with the Android and iOS development environment
- Ability to estimate how long a task will take
- Knowledge of what is a Native Module and how it can be integrated with each native platform.
- Interest in the architecture of front-end applications and can discuss alternative design systems.
- Awareness of the common problems of software development such as memory leaks, cache-validation, open-closed principles.
- Experience and understanding of codebases that leverage X state management solution. (Bonus — this also applies to any code patterns or specific libraries like RxJS)
- Ownership of the product you are developing and you are interested in the intricacies of the product you are developing.
- Experience taking a product through a full product lifecycle: conception, development, and release.

How to Determine What Type of React Native Developer You Need

We've focused a lot on the hypothetical developer you'll hire. Now, let's turn the focus towards you and your needs.

You'll have some specific needs. No two teams are alike. You might lack some specific platform expertise like iOS and Android. You may need a high-level strategist like a senior developer.

Your main job is to identify your deficiencies, and once you have asserted the basic requirements for a React Native developer you can then focus on vetting for what you specifically need.

We have had interesting experiences with finding out about specific traits of a developer that were not explicitly mentioned in their curriculum or Linkedin profile. For example, one developer we interviewed for a React Native role, despite not sharing this on his resume, had extensive experience with writing Android apps with Java. In another case, we found out someone was a natural leader, and given his amazing communication skills and natural instincts, we thought he could be a technical leader.

What kind of role will the newly hired individual play inside your company? What is your company's atmosphere like? Who will this developer work with? These questions are incredibly important.

We often see an excessive delegation of responsibilities where a React Native developer is doing the job of a Scrum Master and team lead due to the lack of engineering leads in the company. There is nothing wrong with this. But, it's best to signal these roles in the interview, rather than the onboarding.

The company hiring needs a clear vision of what they are hiring for and why before they send a contract to a developer.

Some companies like to look for experience in their industry. While this can certainly be a bonus, it shouldn't be a deal-breaker. Someone with experience in the finance field will be oriented to different details than someone with media application experience, but they're both adept developers. Developers with experience in other fields can learn your business fundamentals and also bring more to the table.

Try to identify the kind of team you have inside your company and what kind of developer you need to hire. Here are a few examples of typical team dynamics and traits:

Structured Team Dynamics

This type of team is usually composed of teams with clear leads, project managers, scrum master, and design team. The developer often ends up having a more constrained role such as finishing tasks to achieve the sprint goals.

Unstructured Team Dynamics

This type of team features more generalists. They often have closer relationships to one another's work and more inter-dependency amongst their tasks. The developer has a high impact on the outcomes but also is way more involved in planning, deciding UI, and architecture.

While we don't have the answers to which kind of developer is the best fit for each role, at G2i we have found that being open about how the work environment allows for the developer to decide if the open role is a good fit for them.

Now that we have detailed what React Native developer does, and what are the core knowledge required for this role, there's an essential question lingering. How do we figure out if a developer has all the traits we're looking for?

Questions to Ask When Interviewing a React Native Developer

What are the things a React Native developer should know and how can I ask them about this? What can I expect them to say when prompted these questions? These are all great questions for a hiring manager to ask.

We won't attempt to gloss over complex answers with simple points. The truth is more than one possible answer is correct, our goal with this exercise is to use a somewhat Socratic method so you can ask the right questions and find the right candidate for the job.

In order to go over the fundamentals of React Native, let's imagine what the start, middle, and end of a React Native project would look like. This way, we can rehearse what kind of challenges a React Native developer would encounter throughout the lifetime of a project, and focus on validating the knowledge of the elements that compose these phases.

Ask basic questions to the candidate. Don't take any information for granted or prime the candidate with any previous knowledge. Developers with experience will shine when you ask them open-ended questions without much windup.

How to Get Started Building A React Native App

A project starts with an initial template. So, you might consider asking the following types of questions.

How can I start a React Native Project?

What kind of insights can we derive from this question?

The first answer might be using the React Native Command Line Tool. They can mention how you can start a clean project by typing react-native init my_project or with npx react-native init my_project. They can raise the point of how the React Native CLI's init command outputs an app without any features and native modules, and thus will require more manual work to set up a launch screen, icons, and possibly any other native specific features your app requires.

Always remember that when you're asking these questions, you're trying to assess the developer's knowledge base, not coach them on the proper response.

Another possible answer is using Expo CLI. Expo provides you with an SDK for React Native which includes native modules, multiple project templates, plus extra configurations to get started fast with your project. He might mention how expo has an app for easy beta testing, plus how it supports Web by default.

Often, developers with more experience will also make points on the flaws of each option. If they don't, you can also ask.

What are the drawbacks of using Expo and the React Native CLI?

No technology is perfect. The React Native CLI does not come with all the required features. You are unlikely to finish your app with only what the React Native CLI outputted, thus it might require more manual work and integration with the native counterparts. On the other hand, while Expo tries its best to be a complete SDK, some teams might complain it is too bloated with features they don't use, and thus results in a large APK or IPA file.

Let’s dig a little deeper into this situation. For instance, Expo has changed its strategy in recent years to actually include two different workflows: bare and managed. Expo's main goal with this change was to allow teams to take advantage of Expo's APIs without all of the bloat, and this can be achieved via the bare workflow.

You don't always start a new project very frequently. However, the understanding of how this process happens can point out to you just about how much practice someone has had with React Native. Initializing new projects is the first thing someone does when playing with React Native to prove out some idea.

How to Manage the Layout of a React Native App

What do you think is the right way to write and compose your React Native components?

Once your project is initialized. The next obvious task is to start writing the app. There’s no one perfect way to write React Native Apps. Different teams have different strategies. But, we have noticed two common approaches we prefer that we’ll call the bottom-up and the top-down approaches to writing React Native apps.

In the bottom-up approach, the team breaks down the screens as reusable components. Then they write tasks to create these reusable components and divide these tasks between the team. The main goal is to have a component library that will be used to compose the main pages of your app. These components must be self-contained, and once done, the developer can proceed to build the pages with these reusable pieces, adding the data integration requirements last.

The top-down approach does not start with the components that compose the screen. It instead starts off with the screens. Each developer gets assigned to an entire page, and they work from the data integration requirements, down to the page components, writing lastly the pieces of components their page requires. They might or not be aware of the reusable components.

Either approach your team takes, there are clearly two distinctive parts to writing the app. The layout and data integration. The latter is mostly indifferent to React Native, and more connected to state and networking logic, which are ubiquitous in every JavaScript and React app. The former is a base of React Native apps, which we recommend you validate your developer feels comfortable with.

How is layout managed in React Native?

The keyword you should hear after asking this question is Flexbox. Under the hood, React Native uses a library named Yoga, which offers a simple and intuitive layout API for multiple platforms. Flexbot's naming convention and ideas are clearly connected to React Native. And on React Native's documentation, it says the layout is traditionally done using Flexbox.

Another main difference developers who have worked with the Web will often mention is the difference between React Native's flexbox and the Web's. The default flex-direction is different. The default main-axis on the web is "row", while on React Native it is "column". This is a common source of confusion Web developers have when starting out with React Native.

You can ask about styling. React Native allows you to style its components using the style prop. The StyleSheet is an API provided by React Native for you to construct the style objects. It provides multiple methods and properties for you to use like create, flatten, and absoluteFillObject.

From our experience at G2i, we have noticed most developers who are fluent in React Native won't require any counterproof questions to their overall understanding. However, there are moments where you might just need to rule out that someone is not an imposter. An easy way to do so is by asking a very simple question which anyone who has written React Native UI for more than a month might know from the top of their head.

How do you center a View component on the screen?

There is more than one possible answer to this, unfortunately. But, the most common answer will be using the Flexbox system in your favor. By setting the justify-content and align-items to center should be enough for this question. Plus points can be given for those who can remember that you might need to double-check if the container of the View component is taking the complete space of the screen by setting flex: 1.

How can I handle the layout for different screen sizes?

One of the challenges your app will face for being cross-platform is the different screen sizes. Your UI layouts are required to take this into consideration. The main mistake made when writing layouts with React Native is using fixed values for size and positioning. Thus, a React Native developer who has experienced seeing his app render on an iPhone 11 and on an Android Nexus will know he must use Flexbox with a combination of checks for screen widths and heights using the Dimension API. And again, there isn't a single solution to the problem. But, what we want to hear from the developer is awareness, and what he thinks of the problem. Completely unawareness of this problem can indicate inexperience.

There can be more about layouts. Ideally, your React Native technical interview should be heavily shaped by what the developer can expect to build on your project. For instance, if your company is building TV apps, you should raise questions on how focus can be handled on TV apps.

Once we have initialized the App, done the basic layout, the next challenge the developer will encounter is responding to user events.

How to Manage UX in React Native

While you shouldn't focus on keywords, or buzz terms, I find it hard for someone to explain how we respond to user events in React Native without mentioning onPress, GestureHandler, PanResponder, and React Native's touchable components. At least one of these words should be brought up. The focus here is not to ask: Do you know what the GestureHandler is? But rather, to try to derive from a normal conversation how the developer would achieve something and see if any mentions to these are made.

How do you respond to a user press in a React Native Component?

Despite the different possibilities on how to achieve this, the main one is attaching a callback to the onPress prop of the given component. They could deliberately discourse into all of the press callbacks like onPressIn, onLongPress and etc. But at this point, you can feel confident with their answer.

React Native provides us with multiple components to handle touches, plus a gesture responding system for more complex use cases. The GestureHandler system is something that most are aware of, but very few have actually used it directly since it is a very low-level API. The previous idea comes to show the thin line of vetting someone for real requirements versus specific expertise on things 99% of apps don't require their developers to use.

How to Debug in React Native

Debugging is a vital part of any developer's workflow. Here are a few questions you can use to determine how experienced or adept your developer is at debugging.

Someone added a new feature to the app and it is now crashing when we navigate to the new screen. How would you identify the issue?

Let's walk through a possible answer. Someone's first answer might be logs. If we are storing production logs somewhere in a crash reporting tool like Crashlytics or Bugsnag. With a log, we can determine what caused the crash. You can always follow up with more questions like what kind of crash reporting tools have you used, or what was the worst production crash you have experienced.

The second step might be to reproduce the error locally. Reproduction is an important step to validate the effectiveness of the fix. Despite just running your app after you have made code changes, someone might mention using debuggers like the React Devs Tools, React Native Debugger, and Flipper to help in this process.

Be curious about their opinions around these debugging tools and how they can help. For instance, a debugger can allow you to log and step through your code. React Dev Tools and the React Native debugger are pretty much the same, differing only on the fact that the React Native Debugger is a standalone Desktop app using Electron to wrap the react-dev tools. But, Flipper is a bit different because it gives you more access to the native layers of the app like Networking and the native UI layers.

Someone might mention an issue with the data layer, and how if we used Redux we could use the Redux Dev tool to see when an action was dispatched plus how the state looked at a given moment. There are also GraphQL state managers, which with the GraphQiL tool we can simulate queries and validate responses. You can expect a very varied answer here especially because the React ecosystem has so many ways to solve different problems.

How can we prevent bugs and errors?

Once we have the bug figured out. How can we prevent it from happening again? Why do bugs happen? How do we prevent these unexpected things? Developers are full of opinions. These opinions can tell you a lot.

The preferred answer to this question is testing. Unit testing is a way to assert a behavior will and will not happen, thus when we are introduced with a bug due to unexpected behavior, you can fix it and write a unit test to assert it won't happen again. There are more options. You can add integration tests and end-to-end tests. Another way is to make sure you have a proper release pipeline with QA testers, Beta users before you roll out to production.

You can always dig for more. In the realm of tests, everyone has their own taste of tools and processes. For unit testing, in React Native the default and most loved framework is Jest. However, for end-to-end tests the tools have not yet consolidated, developers might mention Appium, Detox, and even a less known tool like Cavy.

Static Type Checkers are also tools to annotate and constrain code behavior. Today's TypeScript popularity in the JavaScript ecosystem results in many developers preaching its usage as a way to prevent bugs. Ask about their opinions on this practice. There are so many tools to use in this practice, as type checkers when talking about data models, and how having correct type models of your requested network data can help avoid possible crashes due to using undefined data.

Another strong argument of using Static Type Checkers is the proper declaration of the impact surface, you are more confident of how much your changes might impact the rest of the app given how the type check will break if you change a function signature. Nevertheless, this can also be a moment to discuss the drawbacks of add-on type checkers like TypeScript.

Developers might like or detest testing. Some might mention how they love static type tools like TypeScript and Flow. These different tools are trying to solve different problems and knowing how they solve different problems is the foundation for the right decisions in any project.

So what about a basic understanding of the native environments? How can someone build an Android App without knowing anything about its environment? Well, the answer to this is complicated. While React Native and its ecosystem can try to abstract away many of the native parts, they are still a fundamental part of building these native apps. Thus, you should try to derive just how much knowledge they have around the native environment.

How to Manage Native Environments in React Native

What is the default IDE for Android and iOS?

The answer is Android Studio and Xcode. But what are we trying to figure out here? There is underlying knowledge behind knowing the IDEs, we recommend you dig into those, like what kind of pain points can one expect from each of these tools? They are not perfect, and thus, there are different struggles with each. For example, Xcode is a complete IDE that handles code completion, building, compiling, setting up assets, and profile certificates of your app in order to publish it to the app store.

While Xcode is full of features, many developers often struggle with handling the profile certificates in Xcode, it is always painful to get it right and have your environment set-up to build it to actual devices and not just simulators. The accidental complexities introduced by tools are good indicators of experience. Someone who has just gotten started won't have any pain-points with Xcode because they will just not know anything about it. Some Developers, often coming from web backgrounds, might not even know that for some Objective-C code changes to reflect on your app you must rebuild it - this is obvious for native mobile developers and for some who know React Native, but not obvious for beginners.

How are native dependencies handled on IOS and Android?

Let's say we have some framework we want to use in our iOS and Android app, how can it be done? Since this is not just JavaScript code we can't simply NPM install it. This is a common scenario in React Native. While it is possible for someone using the managed workflow of Expo to just use their available native modules, and these don't require any native integration, just a simple import of code. Most React Native developers will have had some experience with integrating a third party native module. If you are looking for someone who is experienced, expect them to mention the tools available for dependency management natively.

For Android the tool is Gradle, it is a build automation tool that controls compilation, packaging for testing, deployment, dependency management, and publishing. In my experience with React Native, I have struggled to understand Gradle errors that are often related to dependency conflicts, where some third party module is using different versions of other packages. I would expect someone with React Native experience to at least mention the existence of Gradle. We don't always expect someone to go deep into this. Gradle is responsible for packaging your Android APK, thus you can expect your React Native developer to have to deal with it to set the minimum Android SDKs and more.

For iOS, Xcode does allow you to manage third-party frameworks in it, but often teams rely on Cocoapods for this job. As a sign of its popularity, Cocoapods became the official way for React Native to handle third party dependencies on iOS in 2019. And when this happened, there were a few developers getting stuck on running their init apps on iOS because the outputted instructions of the react-native CLI forgot to mention that now they needed to run pod install inside the ios folder before trying to install their app onto the iOS simulator.

When the issue is not in the JavaScript code, what do you do?

The last thing you want is a React Native developer giving up on an issue because they are afraid of debugging the native layers of their app. The answers you want to hear are related to using the available tools.

Xcode and Android Studio are rich in debugging tools to inspect your mobile apps natively. You can profile and stop your apps on specific breakpoints, plus analyze memory usage and footprint.

There is a large list of things someone can share with you if they have had experience with debugging an app natively. They might mention using Android's systrace and Android Studio's profiler to narrow down FPS drops, or they might mention Xcode's instruments and how they can profile allocations there.

You don't need to be afraid if someone says they can try to debug it natively using XCode and Android Studio. The truth is cross-platform development is hard and you do need a group of individuals who have different experiences with React, Android, and iOS, in order to be successful at it.

What we like to filter out is complete unawareness of the environment.

The gamut of knowledge derived from experience is an asset for G2i. Our collective has over 200 members with differing backgrounds, in it we incentivize knowledge sharing to unblock our developers when they face different challenges.

Diving into React Native Developers’ Past Experiences

What did you do with React Native?

While you can get a sense of a developer’s experience from their resume, there’s no replacement for hearing the stories from their work, their past projects, and their passions in person.

The tip here is to listen, listen carefully, and allow the person to be the center of attention, speaking in-depth about their experience.

What can you learn from a developer recounting past projects they’ve worked on?

Developers often keep a running mental scoreboard of what worked and what didn’t in past projects. Feel free to ask them about this to gain an understanding of how they weigh success and grow from their experiences.

Ask them about what they did on their last project with React Native. How did they contribute to the codebase? What features did they implement? How did they implement this feature? Do you remember any mistakes you made or maybe things you got really well? What do you miss about that project? What don't you miss?

You can ask more questions and dig deeper into your candidate's experience to learn more from them. Your job is to investigate them and indirectly derive what they know. You can memorize answers to technical questions, but the heart of your ethos as a developer comes out in reflections on your past work.

The Lingering Problems with Technical Interviews

We hope this guide on how to hire a React Native developer can help shed some light on some of the knowledge someone would need to be productive with React Native, and how you could possibly start a conversation with someone to find out.

If for some reason someone is failing to show how much they know React Native, be honest, and tell them that you are having a hard time getting convinced of their knowledge. It won't be uncommon for someone to admit their inexperience with React Native if you open yourself like that during the interview, I have personally experienced this with someone.

At G2i, we narrowed our focus to React and React Native for the past few years specifically because we know how hard it can be to fairly assess someone's skills. And we admit how we might mistakenly reject good candidates, and sometimes hire bad ones. We work very closely with our clients, to make sure when a mistake is made that we can fix it.

We acknowledge technical interviews are not a completely solved problem and we are committed to making it better. At G2i, we believe in the value of code and a standardized metric system.

Finding A Universal Standard for Different React Native Developers

For the rubric, it sets a repeatable and predictable baseline that is founded on industry standards and best practices. Everyone is tested in the same areas so that the score matters and is objective. Without a standard baseline, pass or fail comes down to gut feel and situational.

There is "hidden" talent that you might not normally find because you are overly focused on previous or years of experience.

It's the equivalent of sports teams starting to use more analytics. Sure, it cannot answer all questions around performance and what someone ultimately brings to a team, but you can definitely build a basis for what sets someone apart. The top talent naturally stands out from the crowd analytically.

The Value of Code-Based Assessment

The Scrambled Eggs Approach to Testing

When Gordon Ramsay has a new cook join the kitchen of one of his many restaurants, he asks them to cook a particular dish in order to see how good of a cook they are. Now, given that Gordon is a world-renowned, famous chef you might expect he would have a penchant for dishes like Beef Wellington. You might think he asks new cooks to fix him something complicated and technical. You would be wrong. Gordon evaluates his cooks by asking them to make scrambled eggs.

I know what you're thinking. "Scrambled eggs? Anybody can make scrambled eggs. How does that tell you whether someone is a good cook or not?"

While it's true that almost anyone can make some form of scrambled eggs, only great cooks can make world-class perfect scrambled eggs. Their craft shines when their medium is simple, and understated—like scrambled eggs.

That's what Gordon wants to see in this test. He wants to see new cooks show their skill by making food we assume requires very little skill.

That's the Scrambled Egg Testing Philosophy. It's the philosophy that supports G2i's technical vetting process and it's the philosophy that is apparently completely foreign to everyone else running code challenges and technical interviews.

When most people design a code challenge or a technical interview, they start with a hard problem. You know the type. It typically sounds something like this:

Write a function that can find the 1,003rd value of the Fibonacci sequence. Bonus points: The function should return the answer in less than 1 second.

Now there are two things that make this a hard problem. First, it's conceptually difficult. What I mean by that is it's hard to imagine the solution.

Compare that technical challenge to the culinary challenge of making a scrambled egg. In the culinary challenge, you can picture the problem in front of you, plan a solution, and use your skill to execute on that plan. In the technical challenge regarding the Fibonacci sequence, half your mental energy is spent on just conceiving the problem. You're spent before you can show off your skill.

The second problem with this type of challenge is that generating a solution to the problem is difficult. Even a basic solution requires specialized knowledge and thinking patterns that are out of the ordinary. A perfectly optimized solution requires specialized knowledge of all sorts of technical concepts like Big O time, method performance in your given programming language, and the architecture of the system you're operating within. Most people do not have that knowledge. Even the people who do have that knowledge are probably rusty when it comes to using it. That's what makes generating any sort of solution difficult.

The problem is that asking people to solve hard problems like the one above only measures specialized knowledge, not practical knowledge.

Our code challenge and technical interviews are built around measuring the right things—skill and aptitude. We ask developers to complete a conceptually simple task such as building a trivia app. Then we evaluate their app with rigorous, world-class standards. Almost any developer can complete the task, very few complete it with the level of rigor that we expect. This allows us to identify the craftsmen, the ones who bring their best to every challenge laid before them deploying exceptional skills in the process. It ignores common traps like years of experience or past jobs and focuses on what you can prove in a realistic setting. In our opinion, it is the gold standard of developer testing.

Our Scrambled Egg test has other advantages, too. Since almost everyone can complete some version of it, people rarely feel discouraged or overwhelmed by the problem. They are able to relax and give their best effort. This also means that when they don't meet our standards it feels fair. They didn't fall short because they didn't possess some arcane technical knowledge that they could've googled had they been allowed to do so, they just don't have the level of craftsmanship that we expect yet. We've seen multiple developers take that experience and our feedback from it and apply themselves to becoming better craftsmen. They email us six months later telling us how they landed their dream job thanks to our feedback. That's a win for us. What could've been a negative experience became a positive one and we gained a promoter who will send more developers our way. How many companies can say that about the people that fail their technical vetting process?

If you want to start seeing better results in your technical vetting and hiring decisions then you should apply the Scrambled Egg Testing Philosophy to your process. Yes, it's much harder than just finding a really hard problem for people to try and solve. It requires manually reviewing code, allowing for multiple solutions to a problem, and creating a rigorous set of standards that you use to evaluate the solutions presented. The work will be rewarded with a better hiring process.

Of course, not everyone has the time or resources to design their own Scrambled Egg Test.
That's where G2i comes in.

G2i has developed a keen sense of developer skill. We've evaluated thousands of React, React Native, and Node.js developers, carefully honing our interview skills and constantly improving our vetting process to ensure our clients are working solely with the cream of the crop developers.

G2i pairs incredible companies with incredibly talented developers

G2i is the only marketplace for specialized developers. We focus on finding and vetting JavaScript developers that have deep domain knowledge in one or two areas. When you hire a React Native developer on the G2i platform, they will have passed a test specifically designed to vet their intricate knowledge of the ins and outs of React Native.

We believe in diving deep into one or two pools of knowledge, as opposed to scratching the surface of multiple areas of knowledge. With a deep knowledge base, developers can hit the ground running immediately. Our litmus test is: "Can this developer make an impact in your codebase in the first week?"

Our vetting is extremely deep. Each developer goes through cultural vetting, language assessment, a code challenge, and a technical interview that are specific to the tech stack the developer wants to work with. For example, if a developer that you hire wants to work with React, they will have already passed our React-specific coding challenge.

We believe in complete transparency in the hiring process. You'll receive technical profiles of each developer that include our detailed assessment of their code challenge and technical interview. We also provide the code they wrote and access to a recording of their technical interview. Learn more about how G2i can pair you with the perfect developer for your project and budget.

G2i

g2i.co