

8 Ways to Debug your Expo — The Comprehensive Guide

Use Flipper with Expo development, Jest for unit testing, React DevTools for debugging, or Chrome DevTools



Abdulrahman Hashem · Follow



11 min read · Jul 29, 2022




Listen




Share

 Faça login em Medium com o Google 



Leandro Bueno de Souza
rock.scrapt@gmail.com



Rayssa Christina
rayssa.rodriguesporto@gmail.com

Mais 1 conta

It's really important for us as developers to have the skill of finding errors and the causes behind them. But sometimes the error may be hard to identify and that requires some additional tools and techniques which help us to understand the context behind these errors and resolve them.

In this article, after setting up our local environment, we'll discuss the different tools and ways to catch errors and identify them starting from stack traces to setting up a full platform dedicated to debugging.

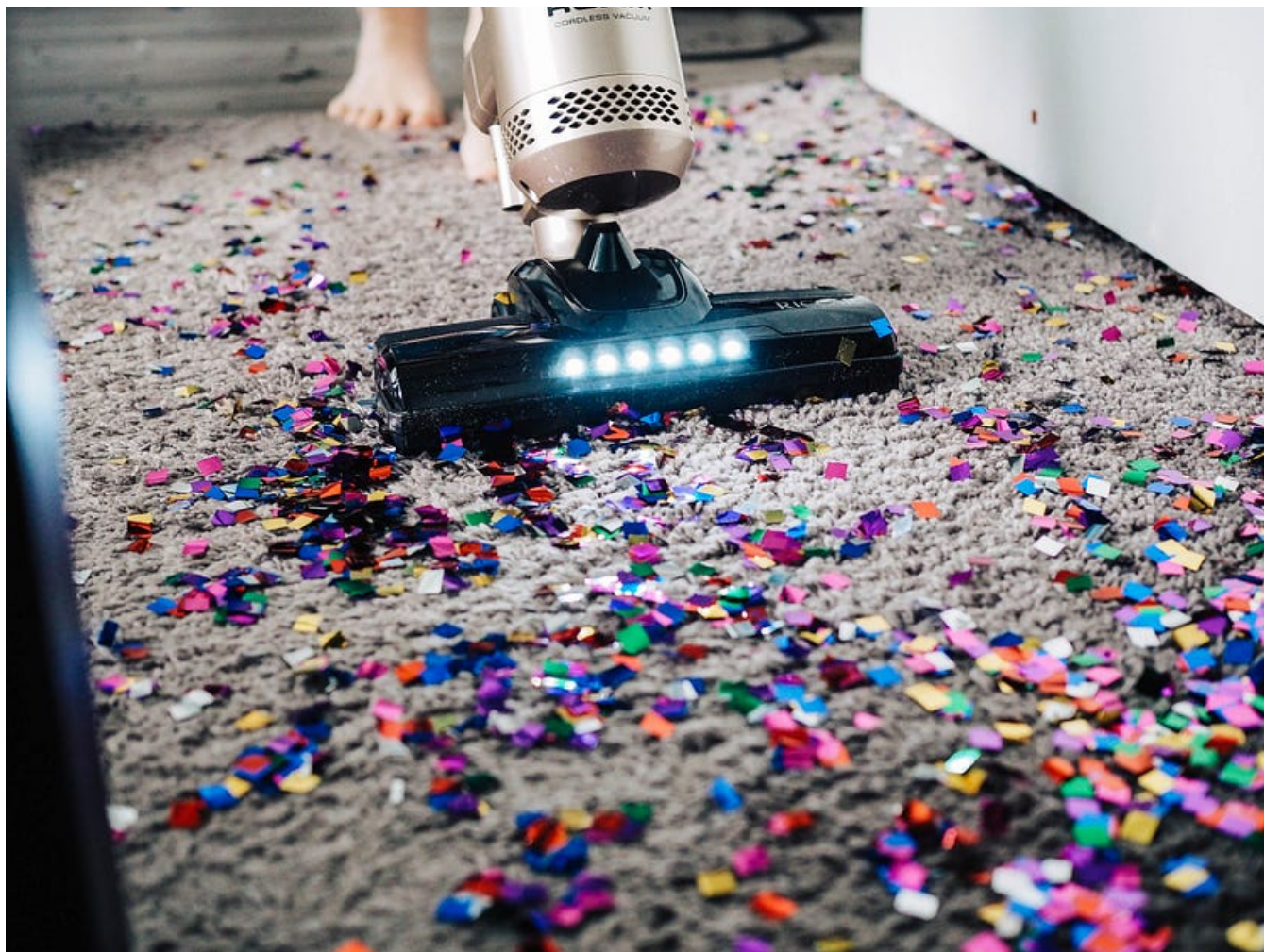


Photo by [No Revisions](#) on [Unsplash](#)

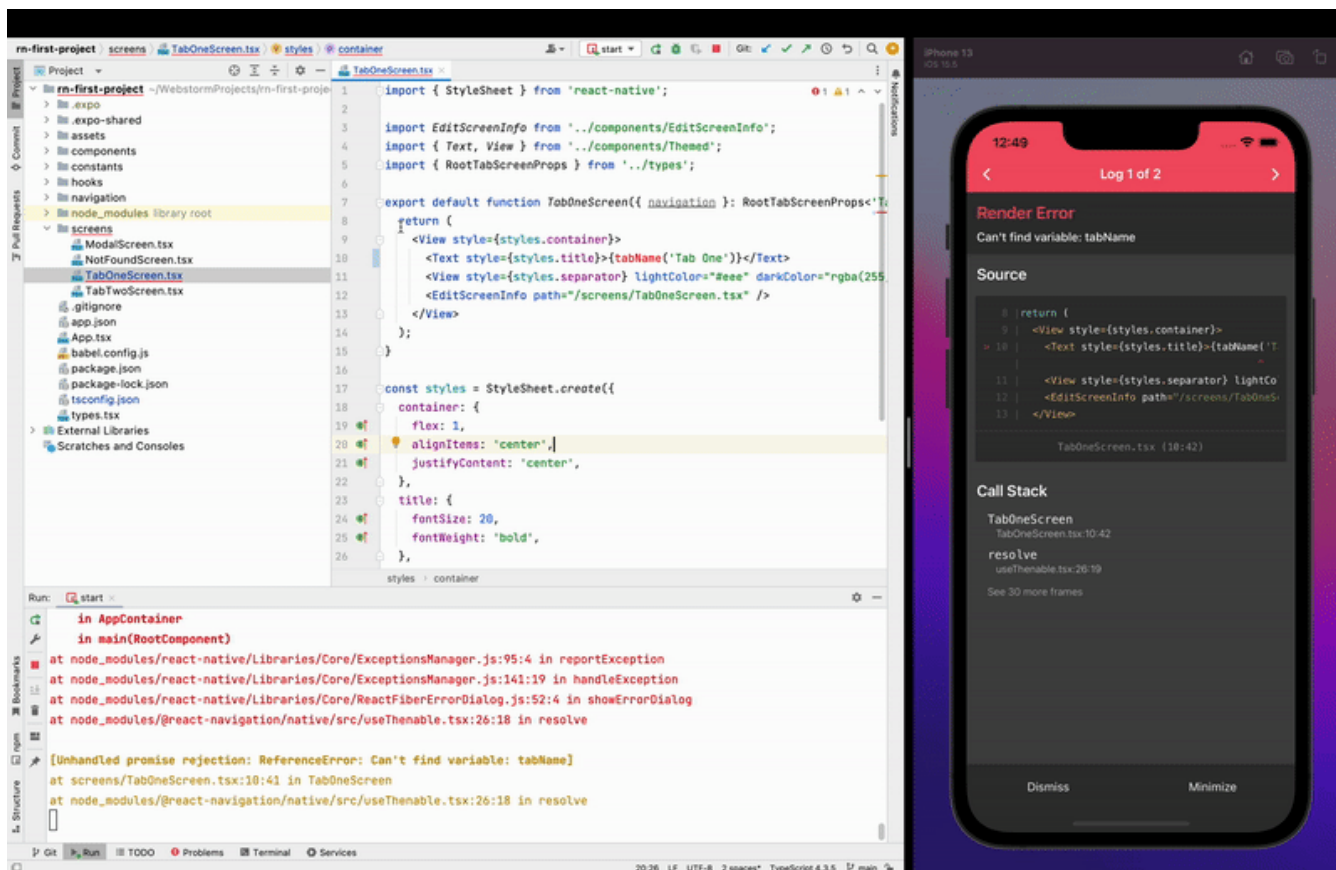
Debugging mobile applications locally is a little bit trickier than web apps which are easily integrated into the browser dev tools. So, let's start checking the different tools to debug React Native app with Expo.

Stacktrace — The easy peasy

Usually, debugging the app locally is pretty easy because most of the time you'll be able to tell exactly what's wrong just by checking the stack trace. In development, it won't be long before you encounter a Redbox error or Yellowbox warning.

- **Red box** errors will show when a fatal error has occurred that prevents your app from running.
- **Yellow box** will show to let you know of a possible issue that you should probably look into now or in the future before shipping the app.

Suppose you forgot to define a callback that is used in your **JSX**, the following stack trace will be extremely valuable since it gives you the location the error comes from. For example, in the following image, we know that the error came from **TabOneScreen.tsx** on line 10 and column 42.



Stacktrace of line and column numbers generating the error

Taking a look at that file, we can see that we are invoking a function that is not declared. Once we add that declaration in, the app will be working again.

However, you can trigger these boxes on your own with:

```
console.warn("Warning message"); // will show the yellow box
console.error("Error message"); // will show the red box
throw Error("Error message"); // will show the red box
```

Sometimes the error message is a little more cryptic which needs debugging efforts to catch, you can follow [a list of steps](#) to take:

- Check Google and Stack Overflow, mostly you are not the first person to run into this.
- Isolate the code blocks that may produce the error.
- If the app is running now, try to investigate more in the isolated code block in order to catch the error behind it.

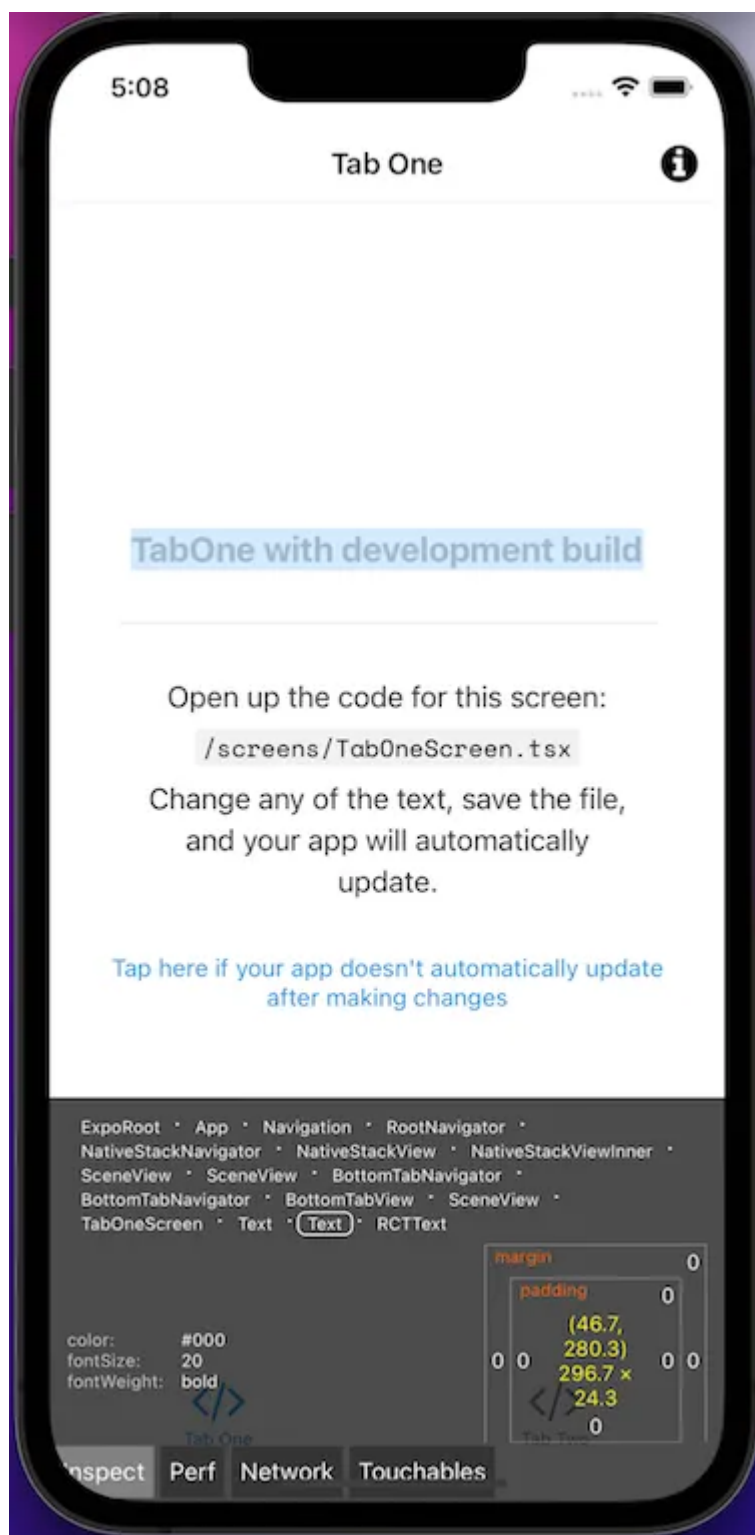
- Use breakpoints and `console.log`s to quickly make sure a certain piece of state is holding the correct value. You can follow along this article to see how to configure remote debugging and breakpoints.

Element Inspector — One click away

A tool that is similar to the inspector that we use in Chrome. You can open the dev menu by doing:

- Physical device: 🖐️ shake it.
- iOS simulator: `Cmd-Ctrl-Z` in macOS.
- Android emulator: `Cmd-M` in macOS or `Ctrl-M` in Windows.

Now try to click on any element on the screen to check its style properties:



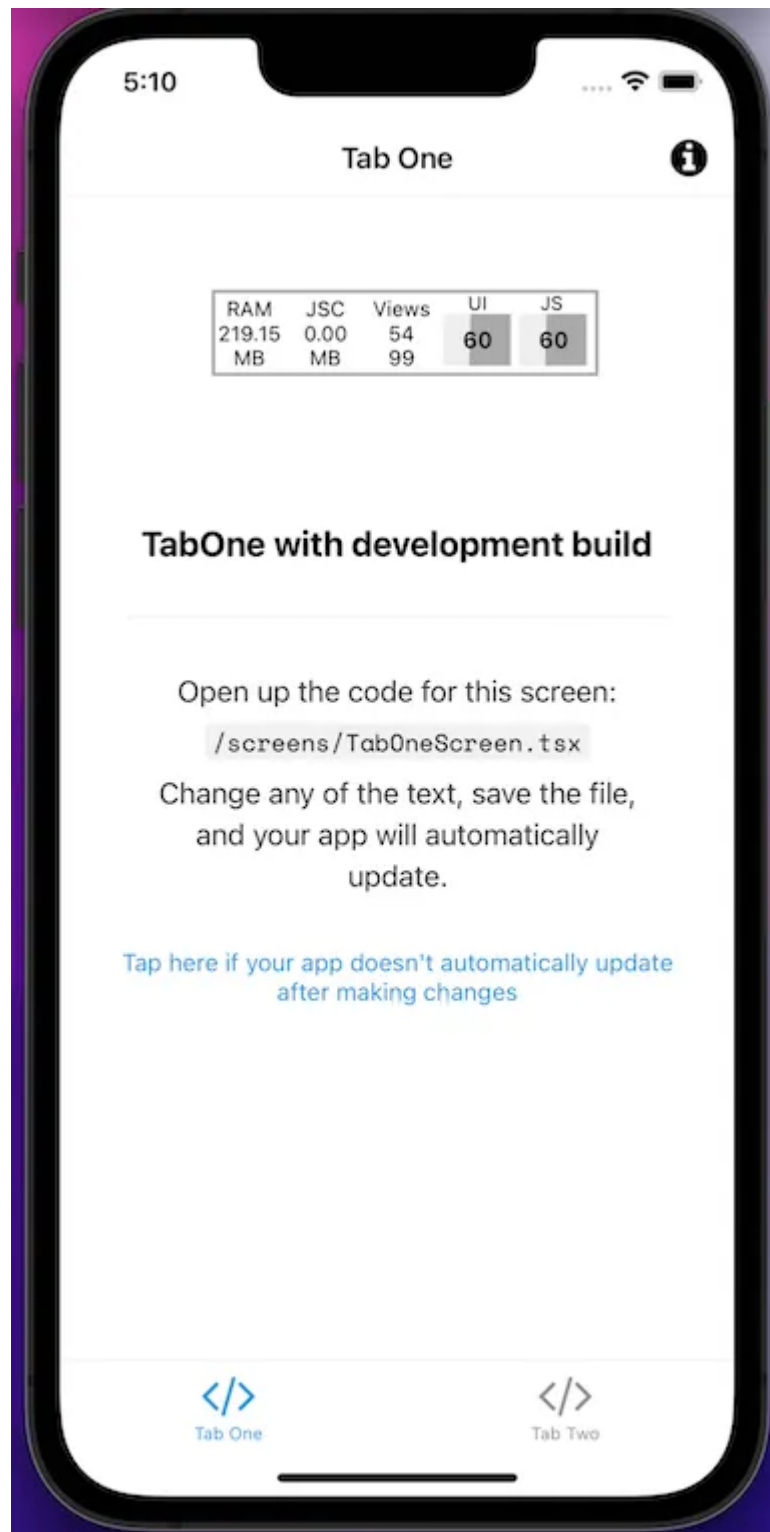
Element inspector in Expo

Performance Monitor — Heap, CPU and Frame Rate

Open the dev menu again and click on **Toggle Performance Monitor** which will show a mini table displaying the following indicators:

- RAM usage of your project.
- JavaScript heap (helps if you want to catch any memory leaks in the project).

- 2 numbers for Views, the top indicates the number of views for the screen, and the bottom indicates the number of views in the component.
- Frames Per Second for the UI and JS threads. The UI thread is used for native Android or iOS UI rendering. The JS thread is where most of your logic will be run, including API calls, touch events, etc.



Performance monitor in Expo

Remote Debugging with Chrome Dev Tools — No Setup Needed

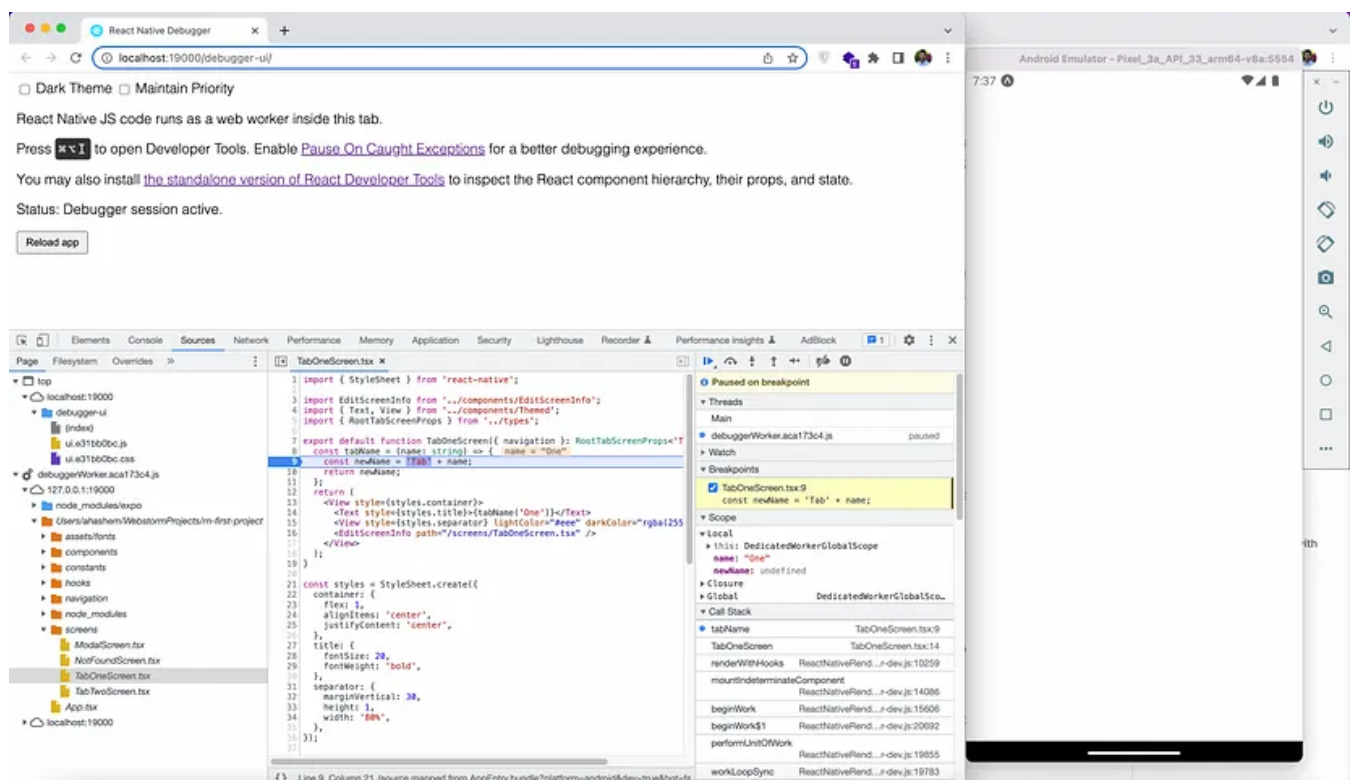
Once you run the project with `expo start`, run the emulator directly by typing `i` if you want to debug on iOS or `a` for Android. This will launch your emulator running your app. If you haven't set up your local environment yet, you can check out [this article](#) for getting everything ready.

When your app is ready, open Expo tools on the running device:

- Physical device: shake it.
- iOS simulator: `Ctrl-Command-Z` (`⌘Z`).
- Android emulator: `Cmd-M` (macOS) / `Ctrl-M` (Windows) (`⌘M`).

Then, choose **Debug Remote JS**. This will open React Native Debugger under <http://localhost:19000/debugger-ui/>.

After that, you can open **Chrome Dev Tools** and navigate to the **Sources** tab. In the navigator, you can find `debuggerWorker/127.0.0.1:1999/path-to-your-project`.



Here, you can navigate through your project files and start debugging. For example, we can debug our `tabName` function by placing a breakpoint there.

Re-open Expo dev tool and reload the app to reach the breakpoint.

PS: if you can't debug the code you are testing correctly, make sure that the

debugger session status is active in the chrome page open.

Debugging with React Native Debugger — All In One Standalone App

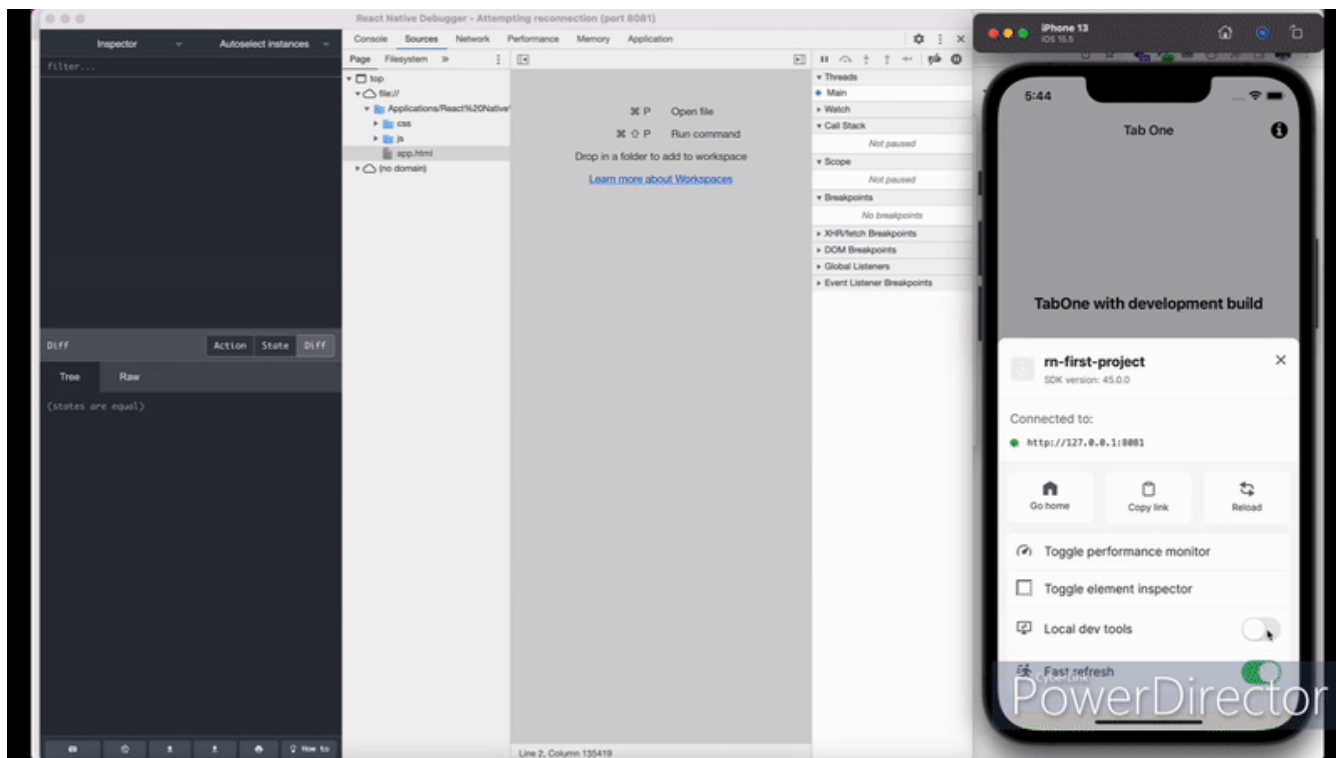
React native debugger holds different tools all bundled into one, including React-DevTools, Redux DevTools, and network request inspection. So probably, this tool is one of the best candidates that can be used while debugging.

For Windows, you can download the `exe` from [the releases page](#).

For macOS, you can install it with homebrew:

```
brew install react-native-debugger
```

- Once installed, fire up the app React Native Debugger and update the packager port which will be 8081 if you run the application with the dev client using `expo start --dev-client`. Otherwise, the port will be 19000 in case you are going to use Expo Go. Personally, I prefer using the development build.
- To configure the port React Native Debugger is listening to, choose **Open Config File** from **Debugger** menu item at the top, update `defaultRNPackagerPorts` to the corresponding port and restart the app. But, if you don't want to change the port permanently, just hit `Ctrl + T` or `Cmd + T` to open a new tab and specify the port.
- Open the app on the simulator and make sure **Debug Remote JS** or **Local dev tools** switch is enabled.
- After enabling it, a new chrome tab will open for React Native Debugger. We have to close this tab in order to free up the port and make it available to React Native Debugger to listen to.



Debugging with React Native Debugger

Debugging with Flipper — Full Platform for Mobile App Debugging

Flipper is an open-source platform for debugging iOS, Android and React Native in a simple desktop interface. It has already included tools to visualize and debug the codebase like log viewer, interactive layout inspector, and network inspector which will help us a lot to debug HTTP API requests.

To make Flipper work, we need to configure a couple of things before downloading Flipper:

- Make sure `eas-cli` is installed globally, or install it with:

```
npm install -g eas-cli
```

- Now, either create a file named `eas.json` in the project root directory or run the following command to create the file for you:

```
eas build:configure
```

- Override the file content with the following configuration:

From the above, we can notice that we have 4 build profiles. Actually, each block under `build` is a build profile that publishes our build through a `releaseChannel`.

If you take `preview` profile, for example, you can notice that we are publishing an internal distribution through `preview` release channel. This build is available internally for our testers to download and check.

Things will get more interesting when looking at `devclient` profile. This profile adds more options and the most important is `developmentClient` which tells EAS to create a Debug build that includes `expo-dev-client` library which is allowed to provide us with tools to help in development.

`ios.simulator` option tells EAS not to create an iOS app, but a bundle that's designed for the iOS Simulator.

- Install development client:

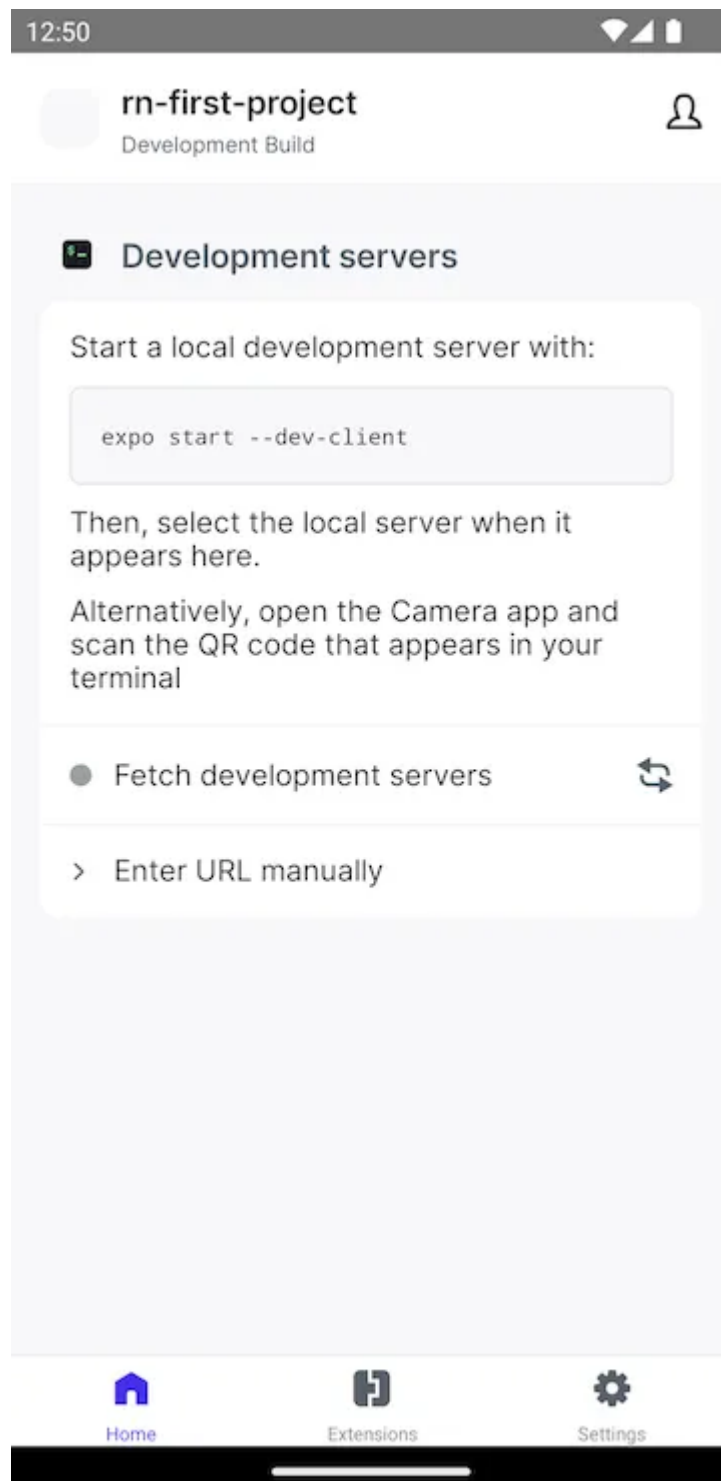
```
expo install expo-dev-client
```

- Add `import 'expo-dev-client';` to the top of `App.{js|tsx}` to improve error messages to be helpful during the development process.
- Run the following to start the build process:

```
eas build --profile devclient --platform android  
// replace android with ios will publish an iOS version to download  
and run using the simulator  
// append --local to run the build on your local machine.
```

Running the build locally will take some time for the first build but will perform faster afterward.

- Once the build finishes, install the app by dragging the produced **apk** and dropping it in the emulator. You can find the **apk** in the project root directory.
- Run the app on the emulator and you'll find a different interface than Expo Go app:



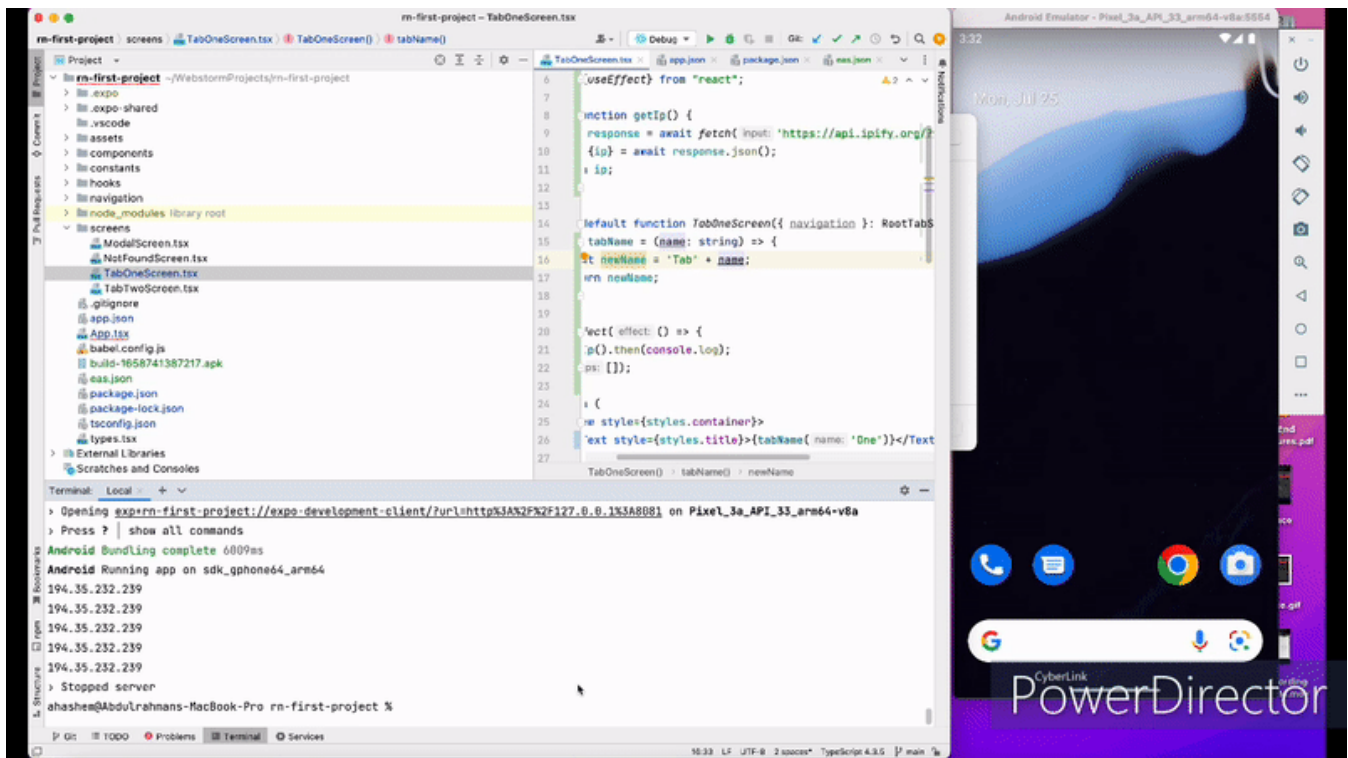
Development build interface on Android emulator

PS: The good thing is that all of the above needs to be set only once. Now that we have the development build installed on the device, we won't have to wait for the native build process again until we change the underlying native code that powers the app.

- All we need now to start developing is to run:

```
expo start --dev-client
```

- Type `i` for iOS and `a` for Android to open the app on the simulator or emulator respectively, then make some changes to the project code and see them reflected directly!



Expo with development build (dev client)

So far, we have replaced Expo Go with the development client running in Expo. To remind you, this is all to get things working with Flipper.

- Open <https://fbflipper.com/> and download the app for your OS.
- Install Flipper and its dependencies by running the following commands `expo install react-native-flipper react-devtools-core` and `npm i expo-community-flipper`.
- Connect our app to Flipper by adding the following in our `App.{js|tsx}`:

- Update `app.json` with the new `expo-community-flipper` as a config plugin:

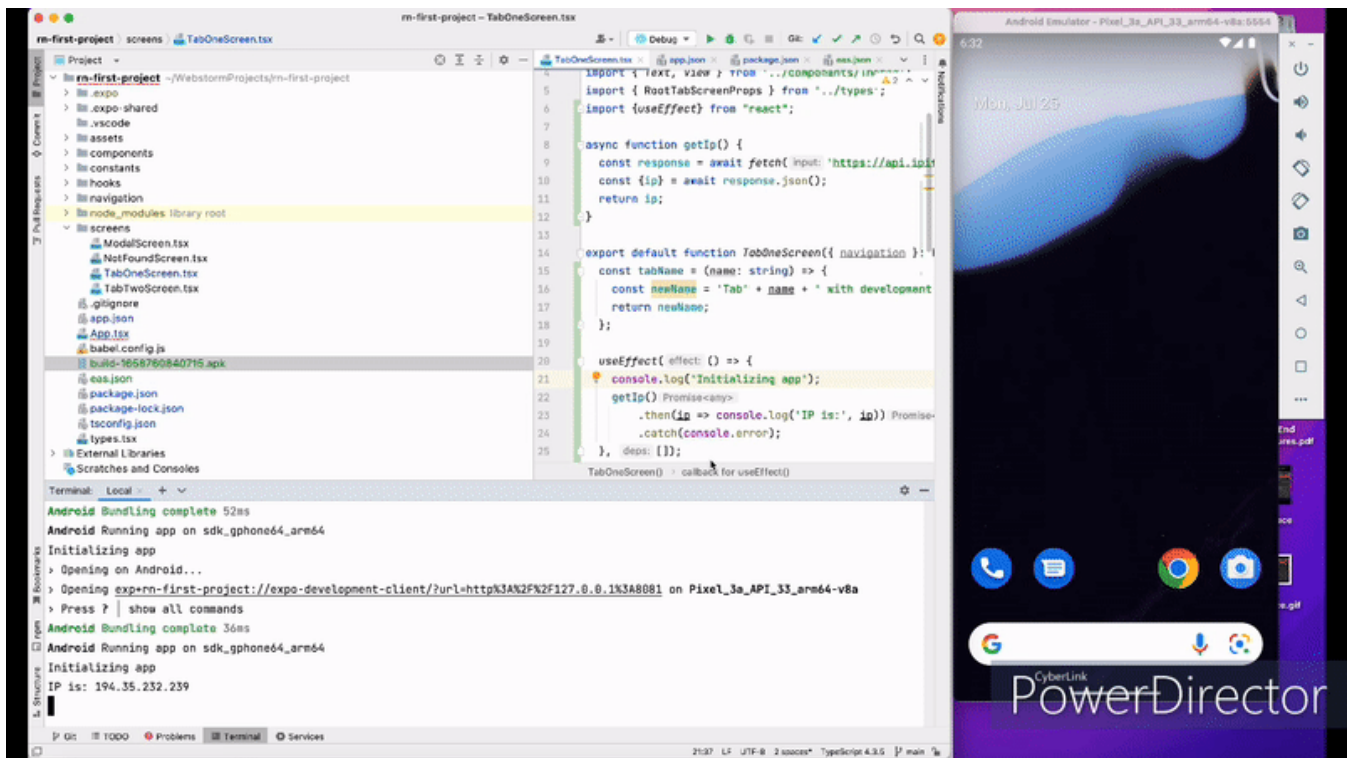
Every time we add a new native module, we have to update our build app and re-install it on the device or emulator/simulator.

- So, one last time, build the app with `devclient` profile locally or on Expo servers:

```
eas build --profile=devclient --platform android --local
```

- Install the app and run the server with `expo start --dev-client`.

- Open Flipper settings and make sure Android SDK path is correct. In case of macOS, show hidden folders with `Cmd + Shift + .` and set the path to `/Users/<name>/Library/Android/sdk` while in Windows the path should be `C:\Users\<username>\AppData\Local\Android\Sdk`.
- Restart Flipper app and Voilà!



Debugging Expo app with Flipper (React dev tools + Logger + Network Inspector)

Now you can easily debug the component tree with React DevTools integrated within Flipper, check your output logs with React Native Logs, and inspect network requests by adding the Network plugin. However, Flipper has many more features and installable plugins.

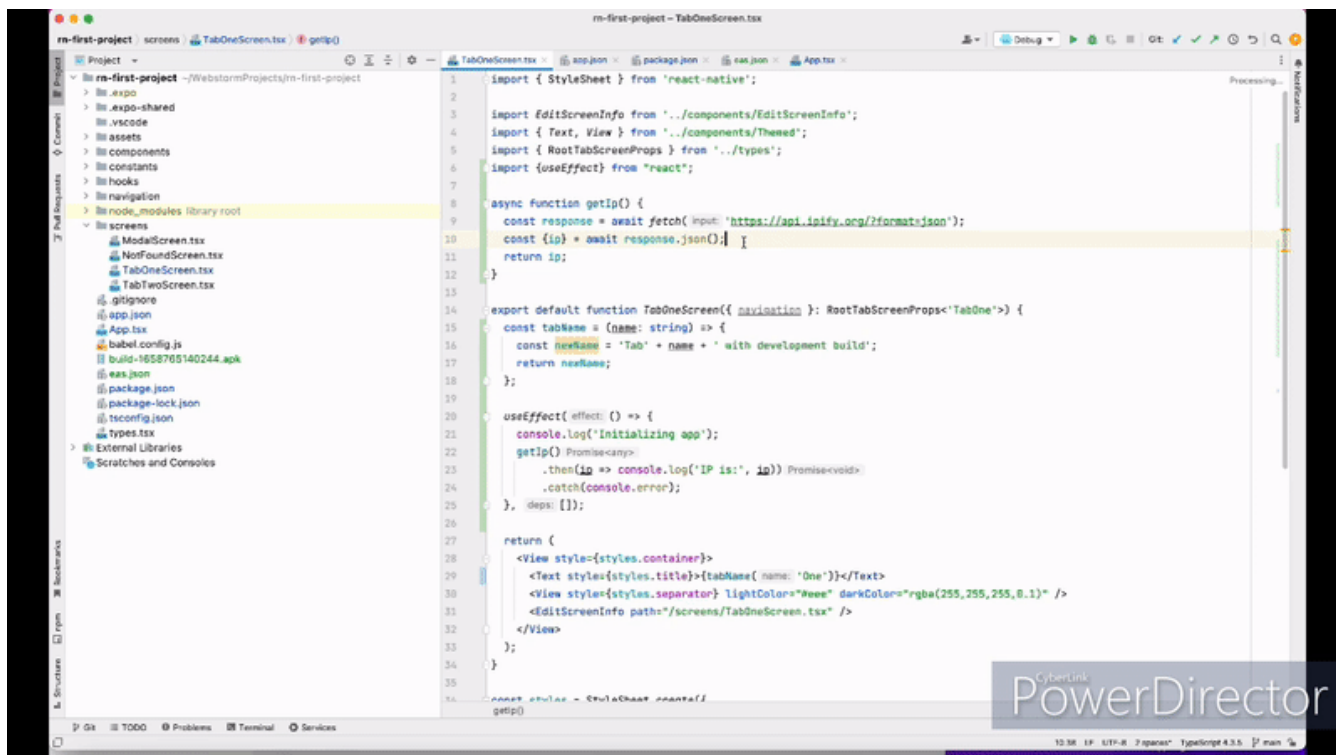
Open Plugin Manager and move to “Install Plugins” tab, wait a little bit for the plugins to load, you can find **redux-debugger**, **react-query-devtools**, **react-navigation**, and many more.

Debugging with Hermes in Flipper

Hermes is a JavaScript engine optimized for React Native and designed for mobile environments focused on startup performance. Using Hermes results in reduced startup time, decreased memory usage, and overall smaller app size compared to JavaScriptCore (JSC).

In order to tell Expo to use Hermes, open `app.json` and add the following to it:

Rebuild the app now and install it again on the emulator, place the `debugger` under the line you are trying to debug, run flipper and reload the app in the emulator once more to load the source code and make it available to Hermes:



Code inspection and breakpoints with Hermes and Flipper

Sometimes we prefer to deal with one work environment or stick with IDE rather than moving between one window and another. So, it would be so useful if we were able to place our breakpoints inside the IDE and stick there while tracing the errors. Fortunately, modern IDEs and code editors support that.

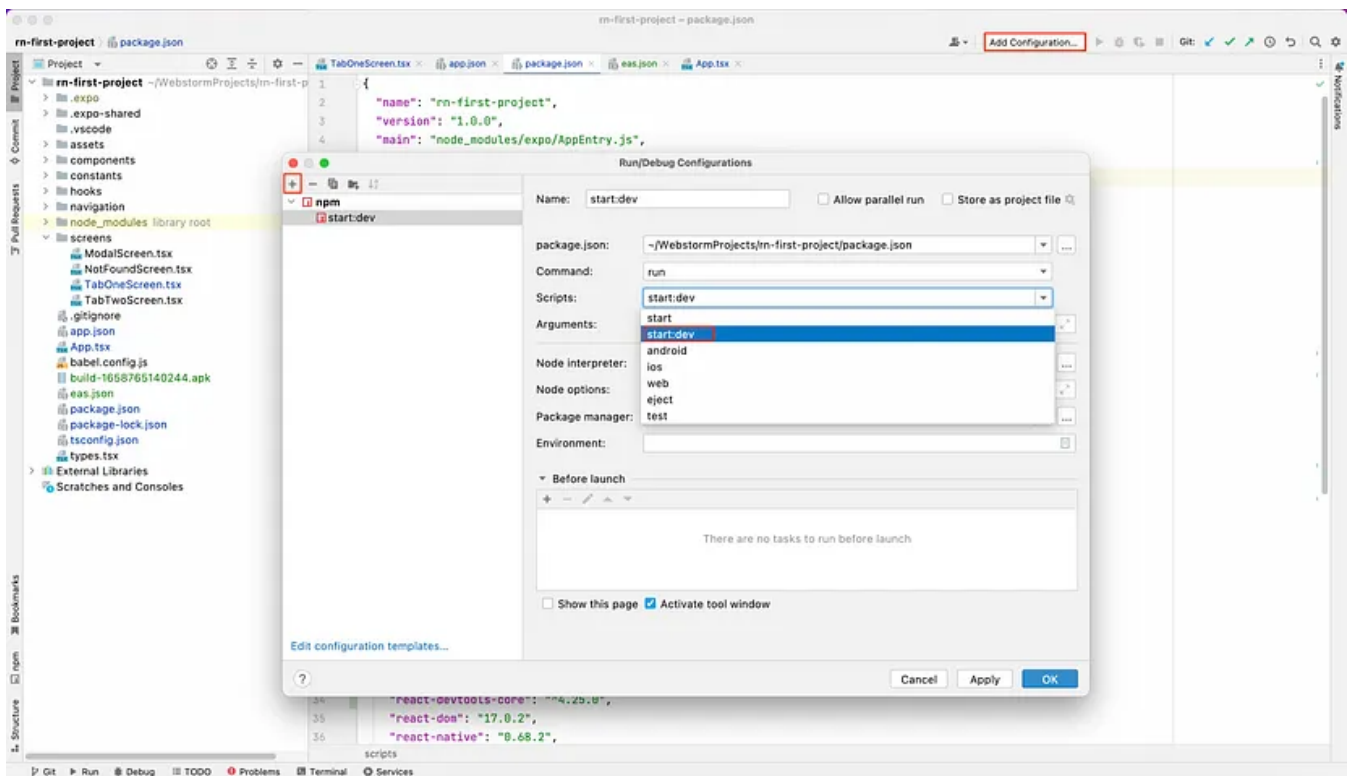
Remote Debugging with WebStorm

To catch our break points in the IDE, we need to configure 3 main steps:

- Open `package.json` and add the following new script which will start Expo with the development client:

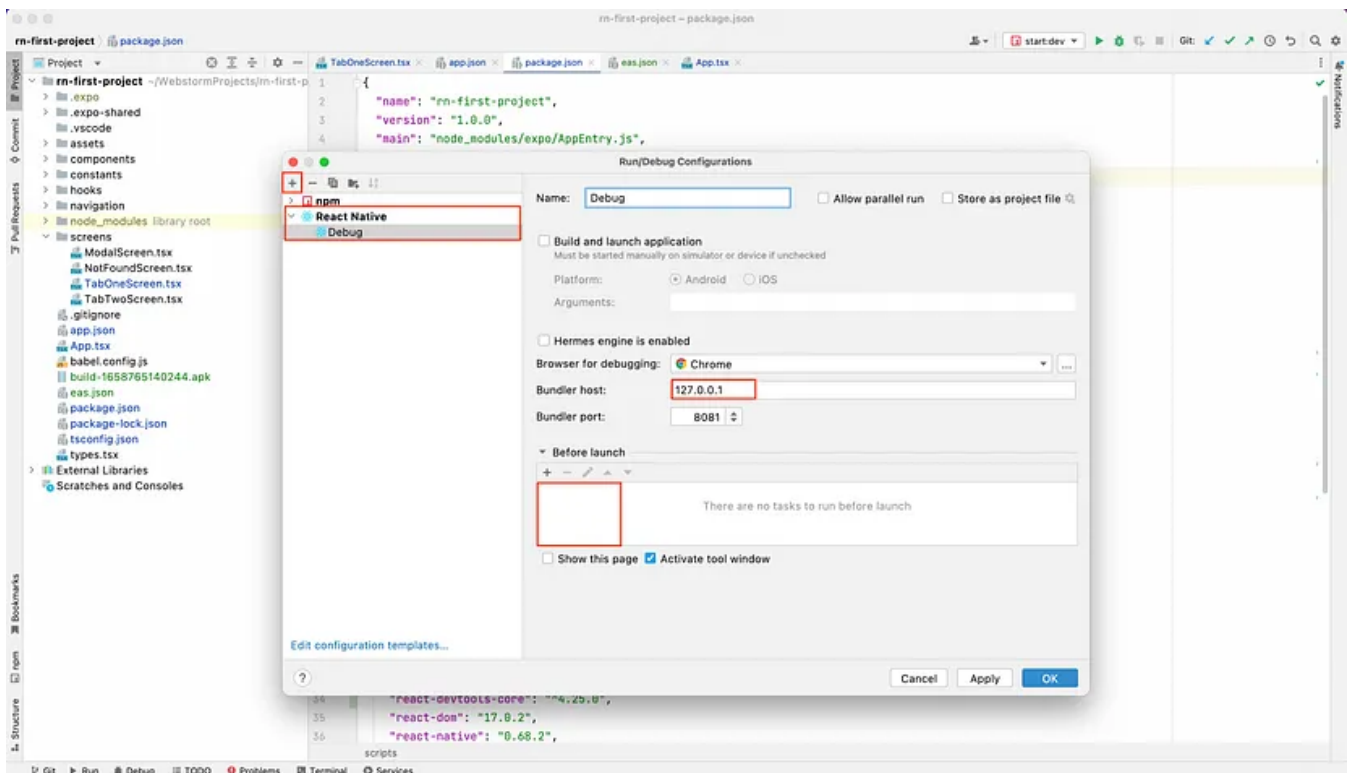
```
"scripts": {
  // ...
  "start:dev": "expo start --dev-client",
  // ...
}
```

- Optional — Add a new npm configuration that will run `start:dev` script for us on green run button click:



Create a new run configuration for Expo with the development client

- Add a new configuration that will attach the React Native Debugger to WebStorm. Make sure to **uncheck** the **Build and launch application** option, update the bundler host to **127.0.0.1** or your local device IP, make sure the bundler port is **8081** and clear all before launch tasks.



Setting up debugging configuration in WebStorm

- Add the breakpoints you want and run the application, then attach the debugger. Make sure to close the debugger chrome tab when Expo initiates it as WebStorm will start one for us:



Debugging Expo with WebStorm

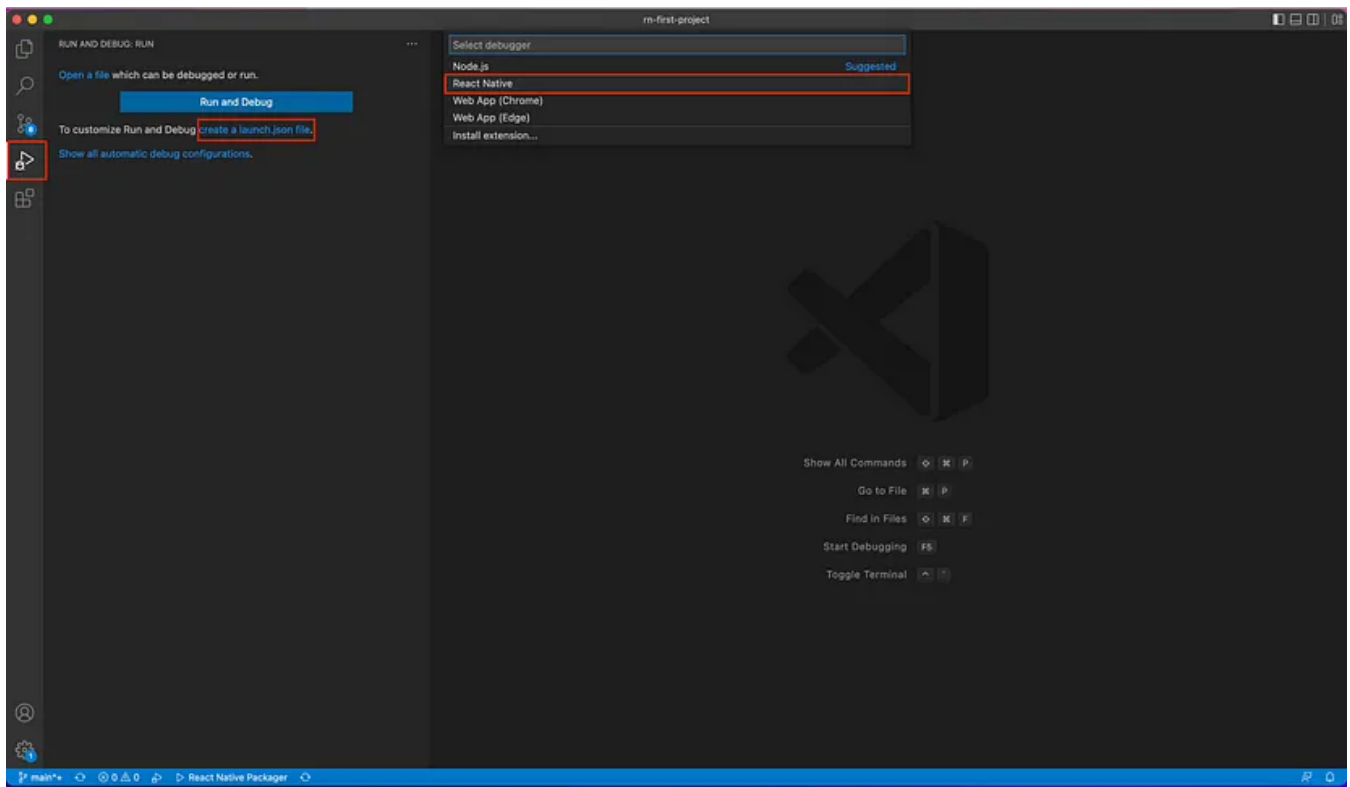
Abdulrahman Hashem

Breakpoints with Expo dev client and WebStorm

Remote Debugging with Visual Studio Code

If you are a VS Code developer, first make sure that **React Native Tools** extension is set, this extension provides a development environment for React Native projects.

- Next, we need to create a debug configuration for VS code by creating `launch.json` file. On the left-hand side, choose `Run and Debug` then click on `create a launch.json file`, choose `React Native` and choose whatever option after that as we are going to change the file manually.



Create a debugging configuration file (launch.json) in Visual Studio Code

- Update the file with the following configuration:

Open in app ↗

Sign up

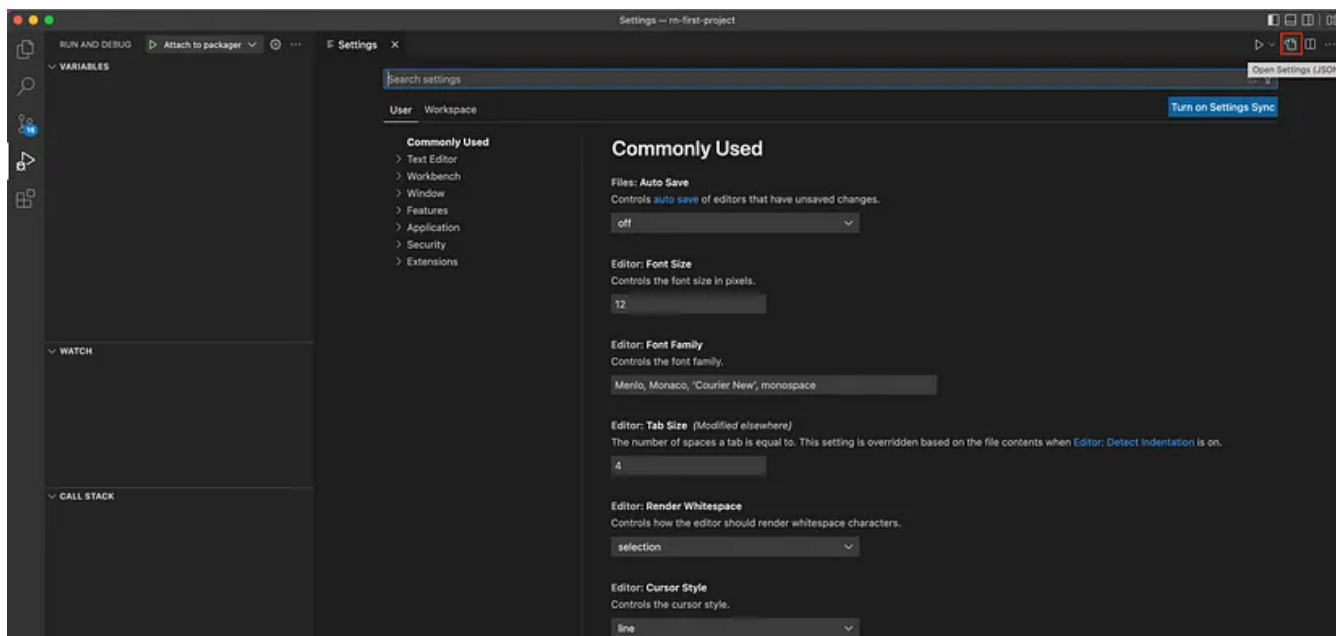
Sign In



Search Medium



- Open VSCode settings and click the icon which opens `settings.json` :

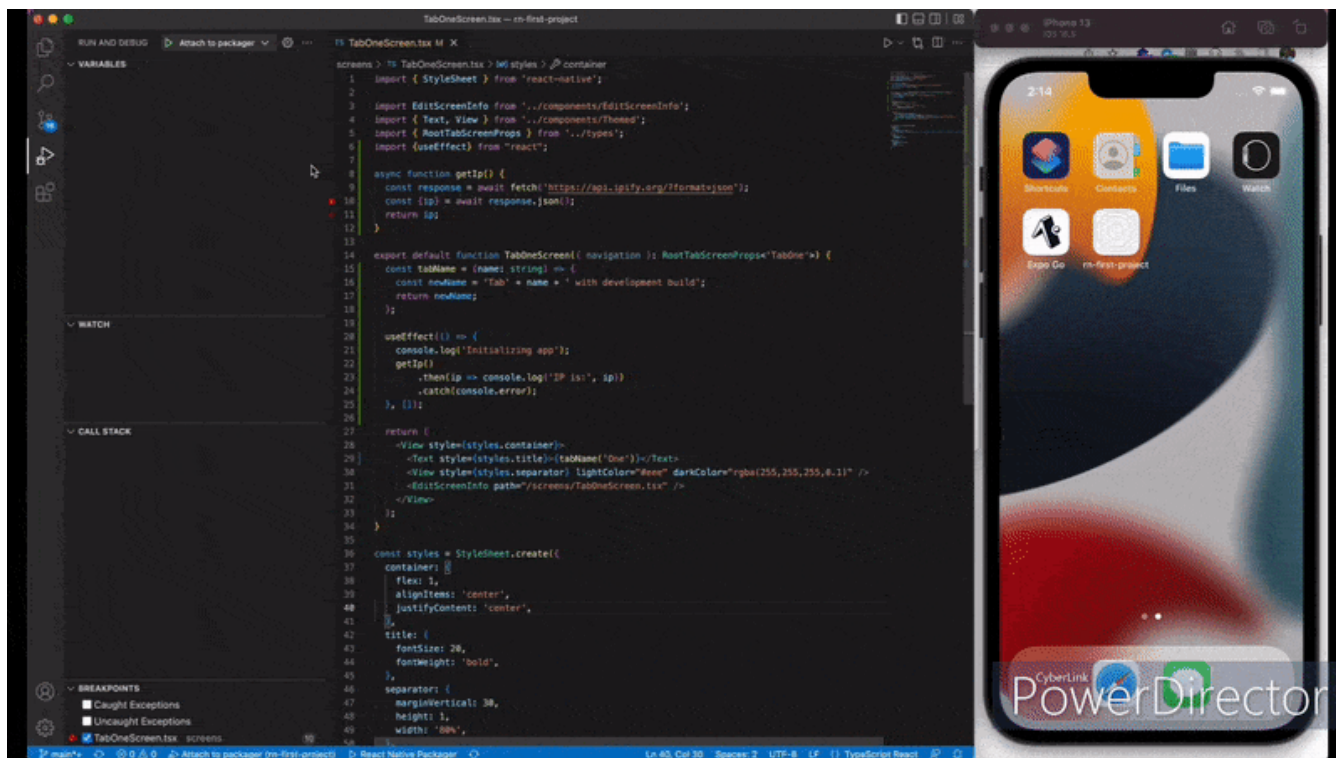


VSCode settings.json file

- Append the following option to `settings.json` or paste it if the file is empty. The port **MUST** be the same port we added in the configuration of `launch.json` :

```
{
  "react-native-packager.port": 8081
}
```

- Set the breakpoints and open **Run and Debug** in VSCode, choose **Run npm start** then run it.
- Choose **Attach to packager** configuration and hit run again.
- The final thing is to open the dev menu in the simulator and enable local dev tools.



Breakpoints with Expo dev client and VSCode

Conclusion

Of course, you don't have to use all of the above tools to debug your app.

So depending on your environment and your preferences, you may combine multiple approaches to build your own optimal debugging experience.

If you have more ideas and thoughts, please share them in the comment below!

GitHub Repo URL: <https://github.com/Abdulrahmanh95/rn-first-project>

React Native

Flipper

Debugging

Code Editor

Expo



Follow

Written by Abdulrahman Hashem

37 Followers

Passionate about JavaScript, web development and front-end frameworks, Angular, React and Vue. Check my portfolio at abdhashem.com.

More from Abdulrahman Hashem



 Abdulrahman Hashem

Angular Reactive Forms—Handling Submission Form State

Displaying fields errors on form submission.

5 min read · Mar 19, 2020

 272  1





Abdulrahman Hashem

My First Week with React Native

Local Environment with Expo — Explore and Set Up

7 min read · Jun 27, 2022



26



1



See all from Abdulrahman Hashem

Recommended from Medium



 Victor Matthew Victor

Expo SDK Upgrading.

As a beginner in the world of Expo, React Native, and mobile app development, I've experienced my fair share of frustration when it comes...

4 min read · May 17



 Softworth Solutions Private Limited

Expo vs React Native CLI

Expo or React Native CLI ? What should i choose ?

4 min read · Jun 27

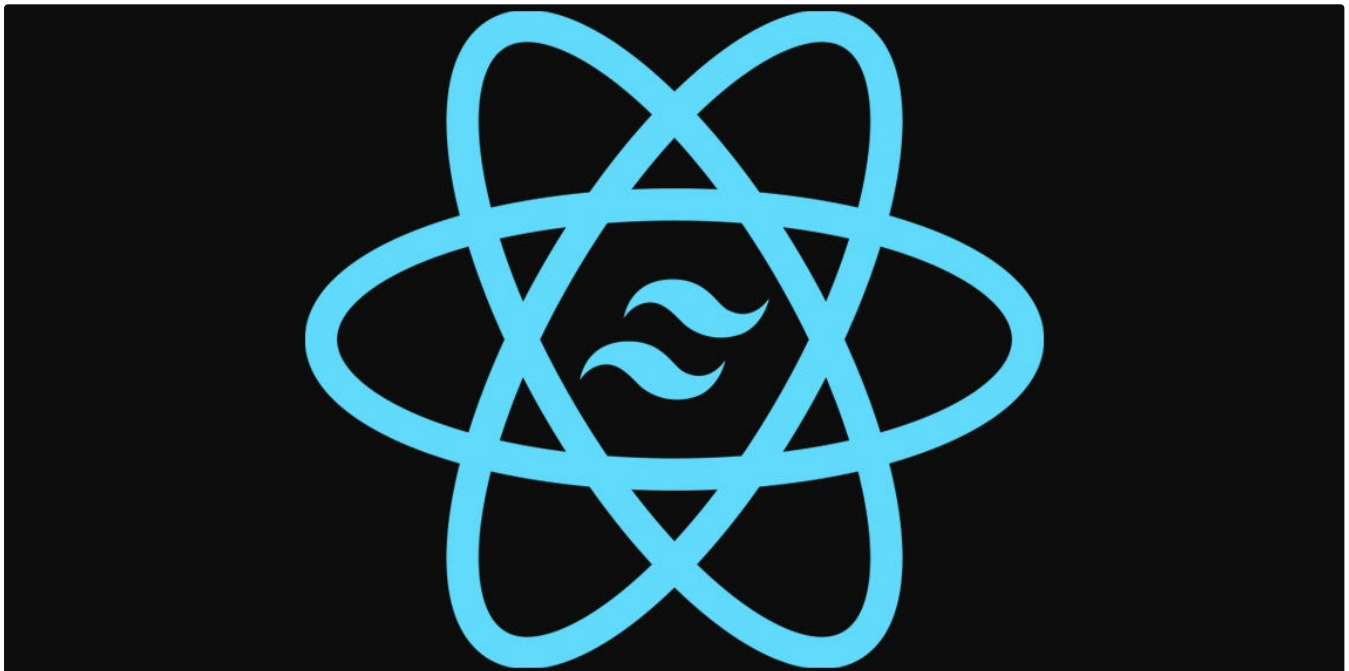


Lists



Stories to Help You Grow as a Software Developer

19 stories · 442 saves



Simple and Short

How to use Nativewind(Tailwindcss) in your react-native application in 2023

3 min read · Jul 26





Diliplohar

NFC Integration Made Easy: Exploring React Native NFC Manager for Seamless Mobile Communication

React Native NFC Manager is a library that provides an interface for working with Near Field Communication (NFC) functionality in React...

4 min read · May 22



8



Tayfun Kaya in Stackademic

React-React Native Optimization

Part 1

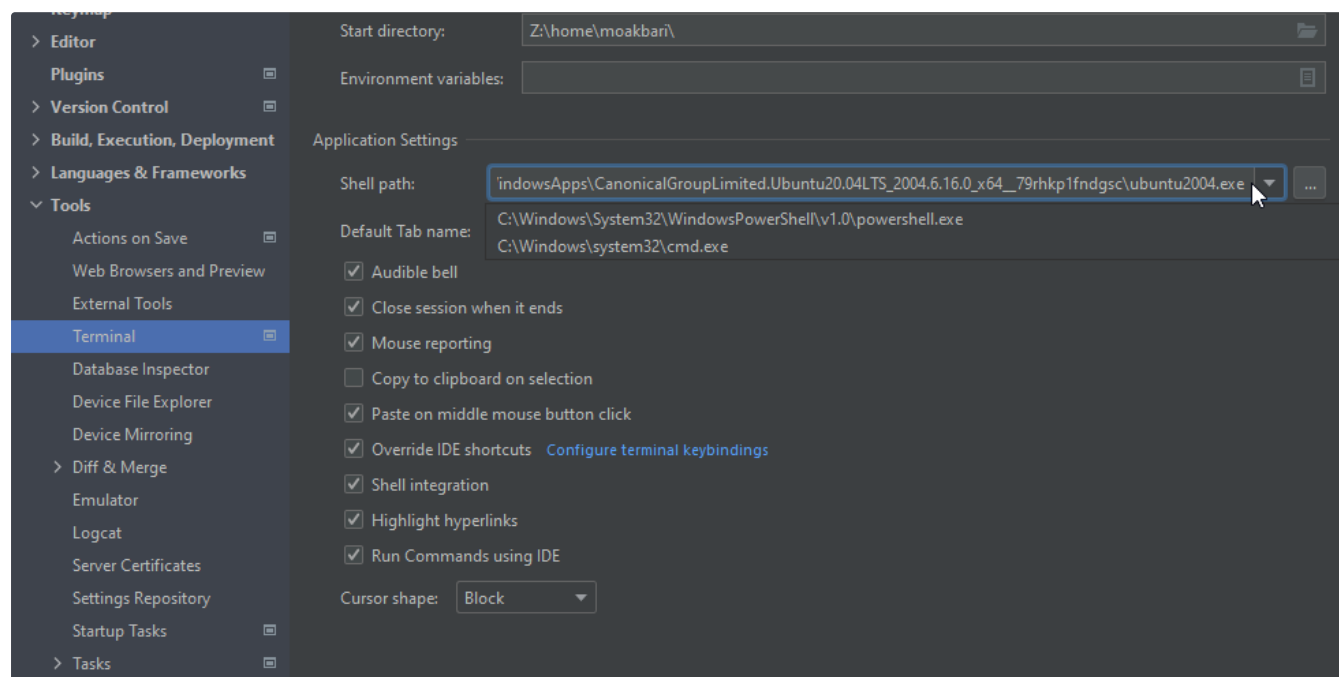
4 min read · Aug 12



347



1



Mo Akbari

Developing React Native with Expo/Android Emulators on WSL2—Linux Subsystem

Hello all, my name is Mo and I'm a software engineer. Recently I wanted to further my mobile development skills, specifically with...

4 min read · Jul 5



11



See more recommendations