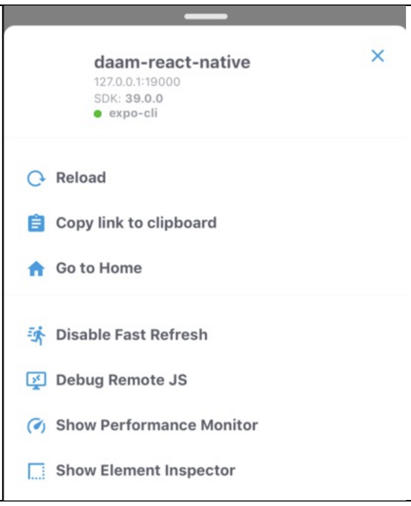
Debugging Lab

It's easy to do "Log Driven Development" (dumping console.logs all over your codebase), but it's not all that efficient. With a bit of extra effort, you can upgrade your debugging experience with better tools.

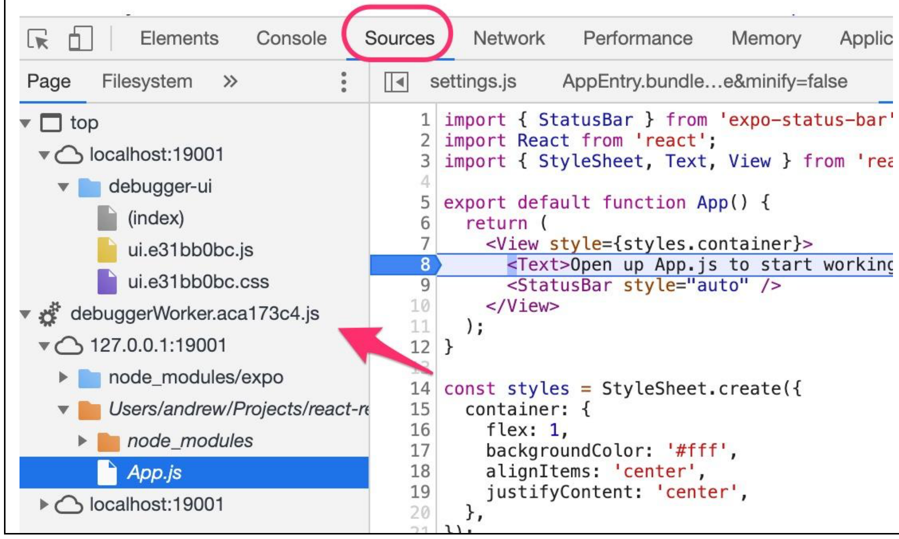
Opening the remote debugger

The first tool is provided out-of-the-box by expo and React Native.

1. Run your app in a simulator.
2. Open your developer menu in a simulator and choose "Debug Remote JS". This will open a new tab in your browser.
3. Now go to App.js and add some console.log statements. Once the App reloads, notice how they appear in the window.



Adding breakpoints



```
import { StatusBar } from 'expo-status-bar'
import React from 'react';
import { StyleSheet, Text, View } from 'rea

export default function App() {
  return (
    <View style={styles.container}>
      <Text>Open up App.js to start workinc
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
```

4. Next, let's try adding some breakpoints. Open up [Sources] > [debuggerWorker.js] > [127.0.0.1] > [your project] > [App.js]
5. Set a breakpoint anywhere you like.
6. Run your app.

Let’s practice writing some React hooks to simulate loading data when the component mounts. It should initially say “Data is: loading” and then flip to “Data is: loaded” after 2 seconds.

7. Go ahead and change App.js to say this:

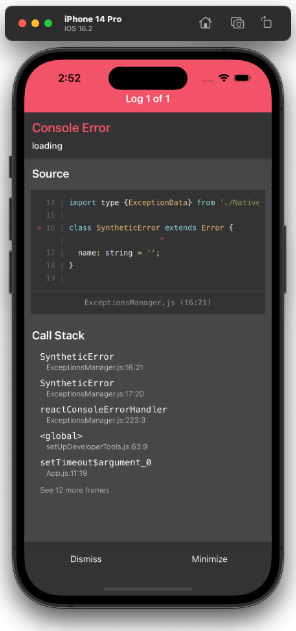
```
export default () => {
  const [loading, setLoading] = useState(true)
  useEffect(() => {
    setTimeout(() => {
      setLoading(!loading) // Toggle from true to false after 2 seconds
    }, 2000)
  },)
  return (
    <View style={styles.container}>
      <Text>Data is: {loading ? 'loading' : 'loaded'}</Text>
      <StatusBar style="auto" />
    </View>
  )
}
```

8. There’s a couple of trivial bugs in this code. Figure them out by using breakpoints.

LogBox

React Native provides a LogBox to communicate with us developers. Let’s try it out.

9. Edit your app. Put in a few console.warn and console.error statements in your app.
10. Run and test. Notice how it kindly indicates which component the message is coming from. (Believe it or not, that wasn’t always a thing. Debugging in React Native used to be… rough.)



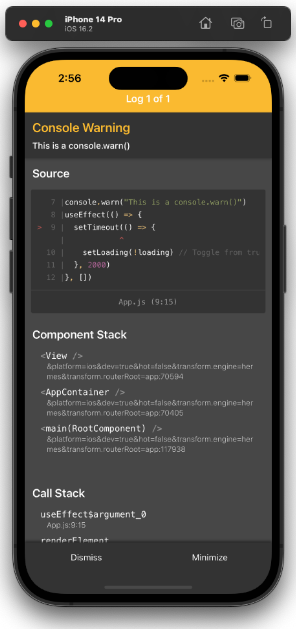
Console Error
loading

Source

```
  14 | import type {ExceptionData} from './Native
  15 |
> 16 | class SyntheticError extends Error {
     |                     ^
  17 |   name: string = '';
  18 | }
  19 |
ExceptionsManager.js (16:21)
```

Call Stack

```
SyntheticError
  ExceptionsManager.js:16:21
SyntheticError
  ExceptionsManager.js:17:20
reactConsoleErrorHandler
  ExceptionsManager.js:223:3
<global>
  setUpDeveloperTools.js:63:9
setTimeout$argument_0
  App.js:11:19
See 12 more frames
```



Console Warning
This is a console.warn()

Source

```
   7 |console.warn("This is a console.warn()")
   8 |useEffect(() => {
>  9 |  setTimeout(() => {
     |            ^
  10 |    setLoading(!loading) // Toggle from tru
  11 |  }, 2000)
  12 |}, [])
App.js (9:15)
```

Component Stack

```
<View />
  &platform=ios&dev=true&hot=false&transform.engine=her
  mes&transform.routerRoot=app:70594
<AppContainer />
  &platform=ios&dev=true&hot=false&transform.engine=her
  mes&transform.routerRoot=app:70405
<main(RootComponent) />
  &platform=ios&dev=true&hot=false&transform.engine=her
  mes&transform.routerRoot=app:117938
```

Call Stack

```
useEffect$argument_0
  App.js:9:15
renderElement
```