Setting Up A React Native Development Environment

Adrian King

Version of December 2nd, 2018

Introduction

These notes are an update to the earlier document describing how to set up a development environment for the Last Call project. I have split the original project into two major parts: a React Native client application and a .Net Web server (with the database.) This means that you can develop React components without any reliance on Visual Studio or the backend server. The server is now hosted in the cloud (http://www.lastcallforfood-dev.com) and you will be able to access the backend services and database as long as you have an Internet connection.

These instructions apply to the Windows environment with an Android target device. An update will be required in order to document setting up on the Mac or for an Iphone target device.

Note that you can still develop components in React and access the data. React Native is really only necessary when you want to test the component on an actual phone. **However**, the markup language in React Native has changed quite a bit and, although it can be done, the recommendation is **not** to use pure HTML. More on this later.

Please contact me with questions or reports of success or failure in getting yourself set up.

Setup

As before you must first have Node.js installed: https://nodejs.org/en/download/ -- if you've already done this then no changes are necessary.

- The project LastCall is available on the Slack dev-discussion group as a ZIP download. Simply unpack all the files in this archive into a new folder 'LastCall'. This hierarchy of files is a complete React Native app. **(Note that you will need about 500Mb of disk space for this due to the ridiculous – and unnecessary – number of files that npm insists on copying into the project.)**
- Put your phone into developer mode. This doesn't affect the normal operation of the phone, it simply opens up a tunnel to a host via a direct USB connection.
- On an Android phone you do this by finding the 'About this phone' menu option and tapping it seven (yes, seven) times. You'll get a message telling you that your phone is now in development mode.
- Download the Expo app from the Google Play Store or the Apple App Store.
- Connect your phone to your host development machine via a USB connection. Supposedly this can also be done over the local network but this has not been very reliable for me.
- Run the Expo app on your phone.

On Windows, run a command line from the LastCall folder. The simplest way to do this is to navigate to the folder using the File Explorer and then enter 'cmd' in the address bar, a new command line window will open with the current folder set to the root LastCall folder.

The tools involved in the next steps are:

Babel – a tool that takes React code and translates it to pure (old) JavaScript so that the app can run on older (non ES6) browsers.

Metro – a tool that bundles all of the necessary files together for download to the attached phone device.

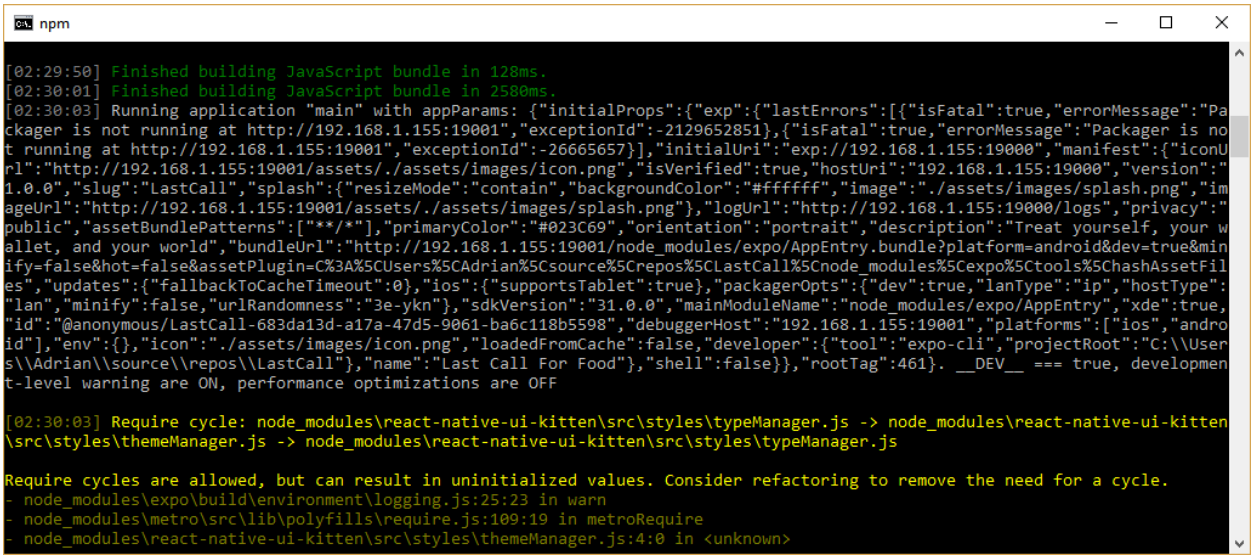
Expo – a tool that downloads the app to the attached device and then monitors changes to the source code (on your development machine) and listens to output from the device via the connected USB so that "console.log('message')" statements in the React code will generate output in the Expo window.

All of this is more fully described in the Expo documentation:
https://docs.expo.io/versions/v31.0.0/introduction/

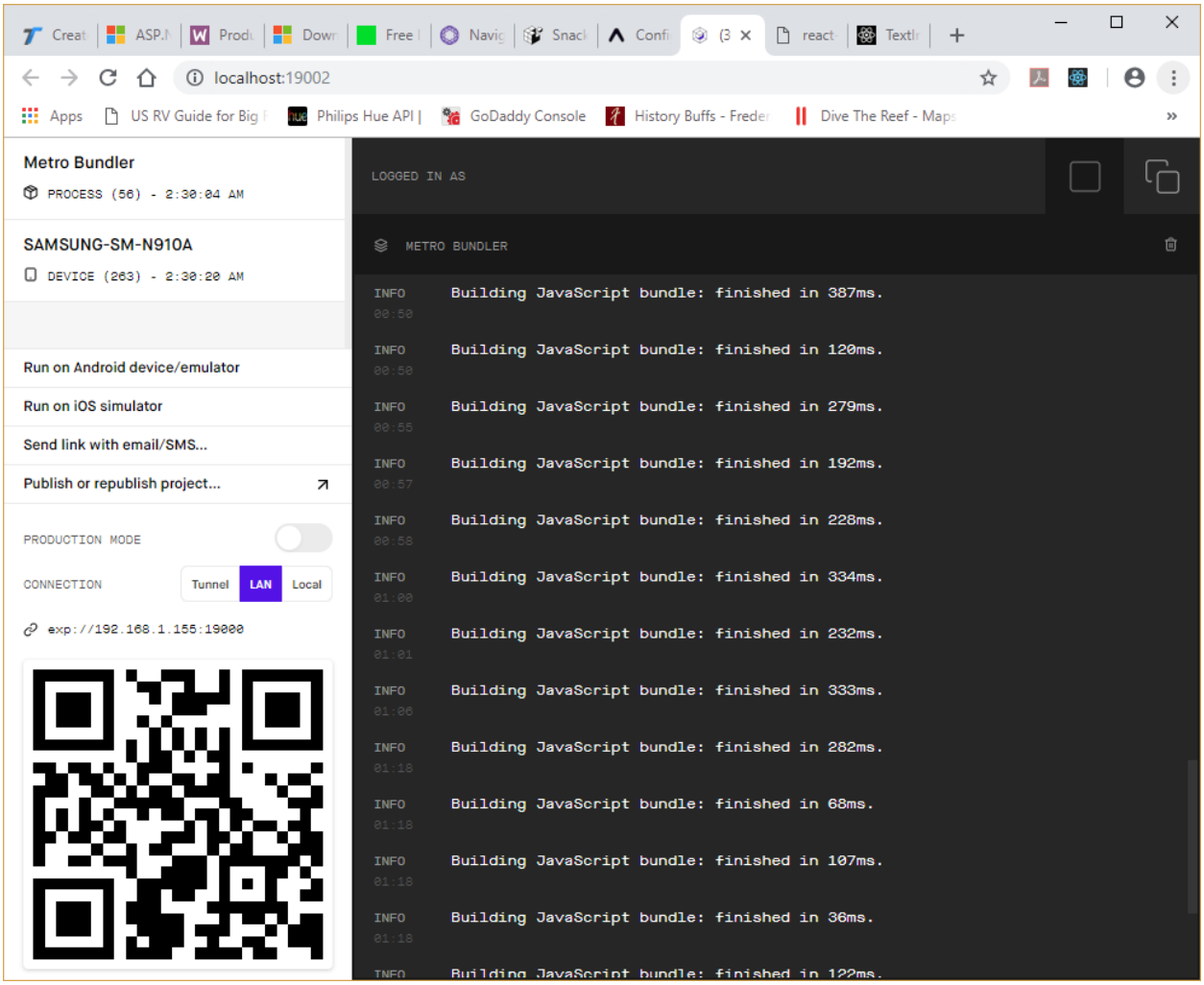
For actual development editing I have been using Visual Studio Code:
https://code.visualstudio.com/download (Linux, Mac and Windows versions) but you may have other tools you prefer. When correctly set up, Expo will automatically reload the app on the connected phone whenever you save changes to any of the files in the LastCall project. Note that sometimes this doesn't seem to work; forcing changes to the App.js file to be saved always seems to work (simply delete and retype a few characters and then save the file.) You'll know that the app is reloading because the screen on the phone will go blank for a few seconds before re-displaying the splash screen.

And, yes, it is kind of a pain until you get it done the first time. After that things work pretty well.

On your development machine, enter the command ‘expo start’, you should see some output like this:

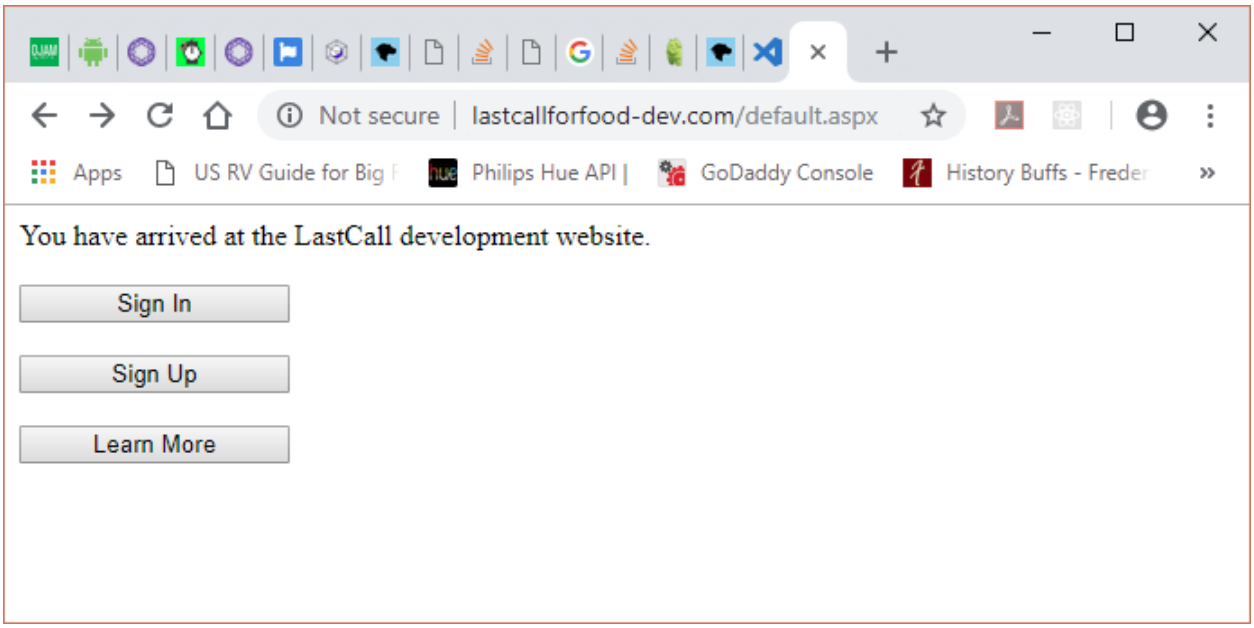


You will also see a new browser window open, looking like this:

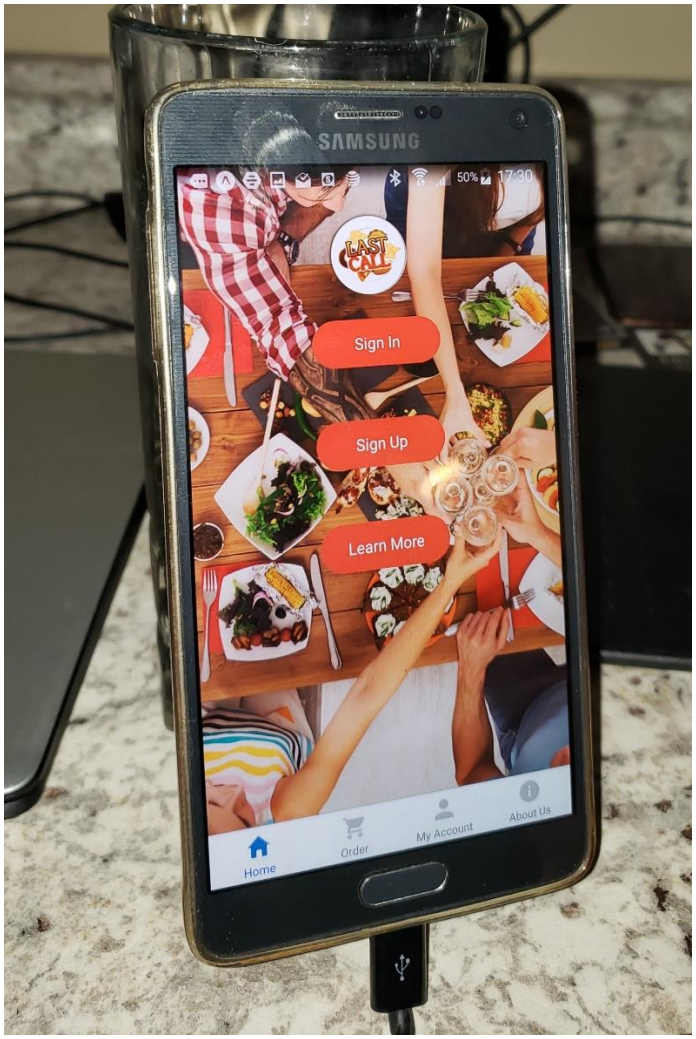


The easiest way to start the phone app (it turns out) is to select the 'Scan QR code' option in the Expo app on the phone and point the phone at the QR code shown. Note: I have had problems scanning the code shown in the browser window, the code that gets displayed in the command line window seems to be more reliable when scanned.

The app will load and start running on your phone. Assuming it has Internet access it will also have access to the LastCall server and database. In order to check you have access, on your phone, browse to: http://lastcallforfood-dev.com/default.aspx and you should see this:



If you have the app running on your phone the splash screen will look like this:



Touch the Sign In button, enter a random username and password, and attempt to sign in. You should see an error message appear ‘Invalid login’ and you’ll see some debug output in the Expo command line window. Be patient during this first try, if the server has gone to sleep due to no activity it may take a while for it to wake up.

If you get this far, congratulations, you now have a complete React Native development environment.

Research Topics

There are several development areas needing more research. It seems to me that assigning these research tasks may allow the team to make better progress than depending upon people to write code and integrate it into the app, though in a few cases some useful code may arise from the research.

Here's an initial list of research topics. In each case, the task would be to research the subject area, make recommendations (supported with a reasoned argument) and, possibly, supply some sample code.

1. The app currently creates and uses CSS styles on each screen. Obviously, we want a single set of shared styles. What is the best way to do this?
2. How do you package a production React Native app for submission to the Apple App Store and the Google Play Store?
3. The development and test environment instructions for an IPhone device need to be verified and documented.
4. React Native defines its own markup language based on a set of components which (eventually) generate HTML. Various styling/markup problems need to be researched and documented. Notably:
    a. What is the best way to incorporate chosen fonts?
    b. How do you do a multi-column layout?
5. There are a number of UI packages available for React Native that extend its base capabilities. These need to be researched and a recommendation made as to which one to use.
6. React Native navigation uses a set of components called, not surprisingly, 'react-navigation'. Although complete, using this library is non-obvious. We need to develop a complete app navigation scenario and lay out the appropriate navigation model for the app. Notably the best practices for switching between screens.
7. What are the best practices for storing and using global data? In particular how should an authorization token be acquired, stored and presented to the server in subsequent requests.