The Development Process Lab

In this lab, we're going to create our initial app and see if we can't get it running in one or more places, a physical device and an emulated one.

1. Create a new React Native app.
```
expo init daam --template blank --npm
```
2. Run your new app by navigating into the new directory and running:
```
npm run start
```
3. If you try any of the options (android, iOS, q), they shouldn't work unless you or someone else has done some installation and setup. Try them out.

Let's make them work

Running on a physical device

4. Whip out your trusty phone or tablet, one that you don't mind installing some software on.
5. Go to the App Store or Play Store and install the Expo app.
6. After the install and setup are complete, go ahead and open the Expo app.
7. Scan the QR code that is hopefully still in browser. If it isn't, hit "q" or if you need to re-start it, expo start again. Then scan the QR code. You should see your app running on your device. Cool, right?
8. Back on your development machine, open your project in an IDE and make a change or two to the main App.js component. You should see those changes immediately on your device.

Running on an iPhone/iPad simulator

If you're not on a Mac computer, you won't be able to do this section. (Thanks, Apple. 🙄)

9. Make sure that Xcode is installed. If it isn't go ahead and install it.
10. Make sure that the simulator is installed. If not, install that also. (Hint: Open Xcode, go preferences-components- simulators - pick a simulator or two - hit install)
11. Go back to your command window where expo start is running. Hit "i" or the Expo dev tools and click "Run on iOS simulator".
12. This should kick off a simulator and run your app in it. You should now be able to see your app running in the simulator on your Mac.
    Troubleshooting:
    - If the simulator never starts, manually open it through Xcode (Open Developer Tool - Simulator). Then try "Run on iOS simulator" again.
    - If the simulator starts but your app doesn't come up, look for the Expo app and tap it.
13. Again, make a couple of changes to the app in your IDE and save. You should see the new version in the simulator.

Running it in the Android emulator

We can't ignore Android now, can we? You can run your app through Android Studio.

14. If you have Android Studio installed, go ahead and open it. On the splash screen you can choose *Configure* (lower right) or open our daam project and hit the *Tools* menu.
15. Choose AVD Manager. You should see a list of emulators installed. If you don't see one, click "Create Virtual Device" and add one.

16. Pick a device and hit the play button (a green arrow to the right).
17. Make sure your emulator is running and hit "a" in the command window or click "Run on Android device/emulator. The server may 'install' Expo on your device if needed. You should see your app running in the emulator.

At this point you may be running your app in three places simultaneously. And changing App.js will result in the app refreshing in all three so you can see how it'll look on multiple devices.

Once you can see your app running in at least one place -- your physical device, an iOS emulator, or an Android emulator -- you can be finished.

Network Requests (Local)

Unlike the web, where your dev environment entirely consists of your machine on localhost, mobile app development has to deal with a little more networking setup.

18. First, we'll practice making a simple GET request to our server from our app. Open App.js and add a useEffect hook that runs on mount to GET from http://localhost:3007/api/films, and console.log the result.
19. **Run it from iOS Simulator**. Notice how this works one might expect; the simulator is on your local machine which has access to localhost.
20. **Run it from Android Simulator**. Notice how this does not work, no network request ever reaches your server (you can watch the logs to verify this).

So what's happening with Android? It turns out their simulator runs in a separate virtual machine, so it's sandboxed without access to your localhost. Fortunately, there's a well-known trick within the community to use a different local API to cut across and reach what is normally run on localhost: http://10.0.2.2.

21. Still on Android simulator, swap out the localhost for 10.0.2.2 and see the network request successfully go through this time.

Of course, it wouldn't work well to have to keep switching out a hard-coded value every time you switched platforms. Fortunately, React Native gives us a module by which we can detect the platform we're running on, and choose a value accordingly. Think of it like a switch statement for your OS platform. Basically, the code will look something like this:

```
import { Platform } from 'react-native'

export const host = Platform.select({
  ios: 'http://localhost:3007',
  android: 'http://10.0.2.2:3007',
})
```

There's a file in `/starters/helpers/api.js` that already has this code. Feel free to copy the file and paste it into your project directory. You can then use it to grab the host dynamically:

```
import axios from 'axios'
import { host } from './api'

export const App = () => {
  useEffect(() => {
    (async () => {
      const { data } = await axios.get(`${host}/api/films`)
      console.log('data', data)
    })()
  })

  return (/* ... */)
}
```

Once again, make sure the network request works on both platforms.

Network Requests (Real Device)

Making network requests work across a real device will be even trickier. Even though it's close at hand, it might as well be a hundred miles away as far as your network is concerned.

With a little luck, it won't be too difficult to achieve. A couple pieces of advice:

A. DON'T do this over VPN, it's a recipe for madness
B. DO ensure both your device and laptop are on the same WiFi network

Since your device isn't the same as your laptop, it obviously doesn't have access to your localhost. Fortunately, your laptop is "generally" discoverable to other users on the same network using your "local IP address" (as opposed to the public IP). If you're on Mac, run the following command into your shell:

```
$ ipconfig getifaddr en0
```

If you're on Windows, you can try doing the same assuming you have a bash shell, or you can follow instructions online like this one (https://www.avg.com/en/signal/find-ip-address).

Your backend server normally likes to run on localhost. You're going to tell it explicitly to run on your local IP as the host. If the "ipconfig" command worked for you, all you need to do is go into your /server directory and run

```
$ npm run server-device
```

If your "ipconfig" command didn't work, you'll need to open up the package.json file in this directory, and replace the line

```
"server-device": "json-server --config json-server.json ./database.json -H
$(ipconfig getifaddr en0)"
```

with this one:

```
"server-device": "json-server --config json-server.json ./database.json -H
YOUR_IP_HERE"
```

just replace YOUR_IP_HERE with your local IP.

If your server is running correctly, you should see resources pointing to your local IP now:

```
Resources
http://192.168.1.2:3007/films
http://192.168.1.2:3007/theaters
http://192.168.1.2:3007/users
http://192.168.1.2:3007/showings
http://192.168.1.2:3007/reservations
```

You're almost there. Now, you'll need to update your URLs in api.js to point to your local IP as well. You can simply set host = your_ip for both ios and android, it should look something like:

```
export const host = 'http://192.168.1.2:3007'
```

Now test it on either an iOS or Android device to make sure it works! Congratulations, you've wrangled the mysterious beast known as networking!

If you weren't able to get it working, or you're stuck behind a VPN or firewall, you can try other solutions like ngrok (https://ngrok.com/) that specialize in making public endpoints that can access your localhost. Even if you don't use it in this case, it's good to know about it as a tool and try it out some time.