Yash Anand's React Native Testing Documentation

Introduction

Hello, I am Yash Anand, a student from KIIT University, working on an assignment for Midas Lab. This documentation will guide you through the process of setting up a React Native application using the Ignite boilerplate, implementing end-to-end tests with Maestro.

Prerequisites

Before you begin, make sure you have the following installed on your computer:

1. Node.js and npm (Node Package Manager)
2. React Native environment
3. Visual Studio Code (VS Code)

Step 1: Setting Up Node.js and npm

1. Install Node.js and npm:

- Visit Node.js official website.
- Download and run the installer.
- Follow the installation instructions.

2. Verify Installation:

- Open a terminal or command prompt.
- Run the following commands to ensure Node.js and npm are installed:

node -v
npm -v

- You should see version numbers for both.

Step 2: Setting Up React Native Ignite Environment

1. Install React Native CLI:

- Open a terminal.
- Run the following command:

npm install -g react-native-cli
npm install -g react-native-ignite
npm install -g react-native-cli

```
npm install -g react-native-ignite
```

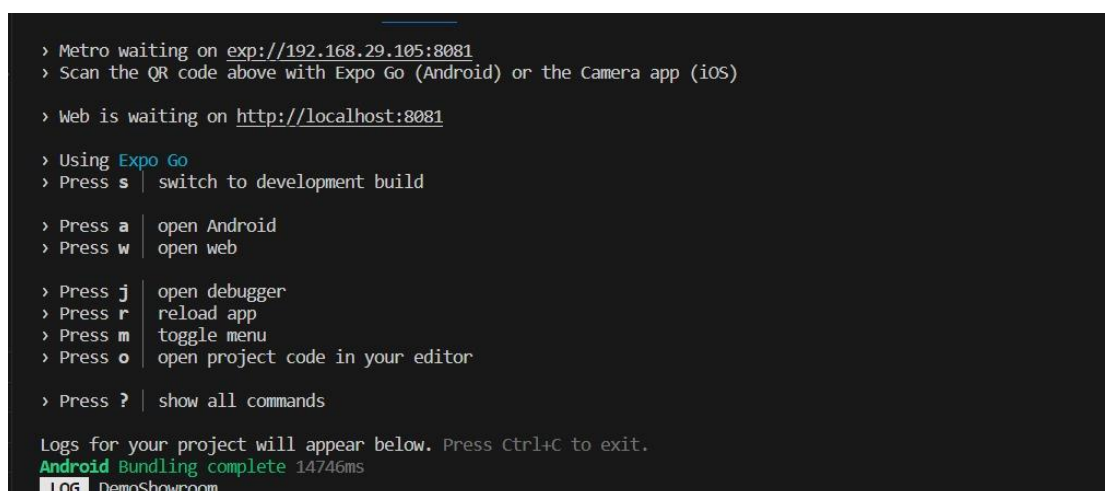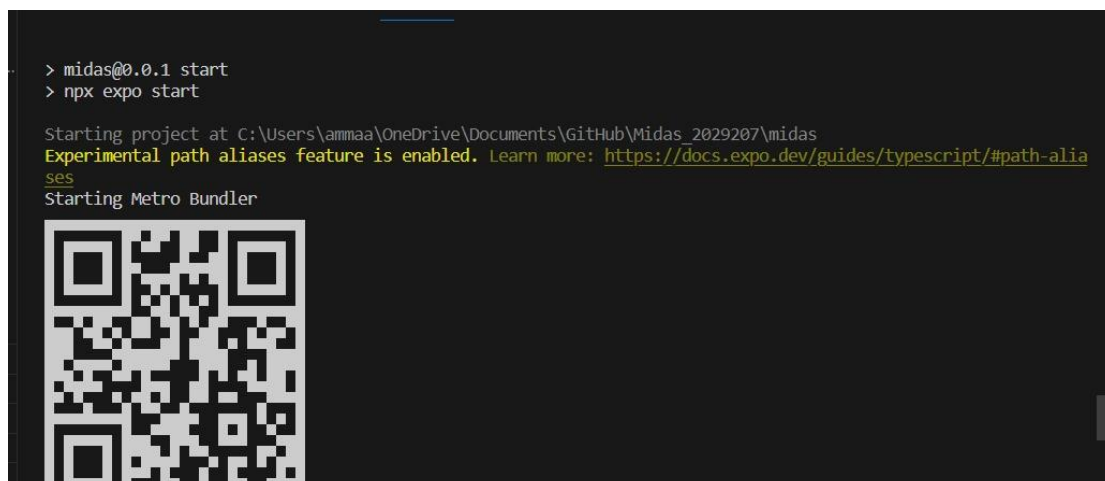2. Create Ignite application:
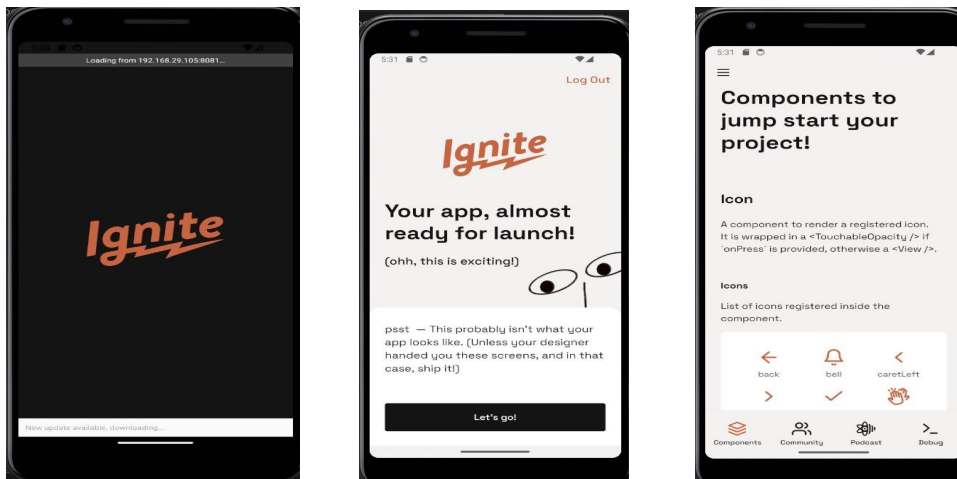
- Ignite new MyApp

```
> ignite new  midas
```

Get into the desired directory and run the following command to check whether the Ignite is connected with your simulator :

npm start

● **Check if in the simulator the following Output is the same**

Step 3: Setting Up Maestro:

1. Install WSL2 for your current Window:

- Open a terminal.
- Run the following command:

  wsl --install

Note: Don't close the PowerShell terminal!

After running the above command, follow through instructions and restart the computer.

Set your Linux username and password (Something that you will not forget).
Run the following two commands to update your Ubuntu system. Enter password when prompted.

  sudo apt update
  sudo apt upgrade

2. Install JAVA:

● After restarting the system, open the Terminal application and click on the dropdown to select Ubuntu. Type in the following command:

sudo apt install openjdk-11-jdk

```
sudo apt install openjdk-11-jdk
```

Step 4: Let's set you up to use Android in your freshly installed WSL2:

● Download the Android command line tools zip file from Android official site. (For the Linux)

Command line tools only

Platform	SDK tools package	Size	SHA-256 checksum
Windows	commandlinetools-win-11076708_latest.zip	153.6 MB	4d6931209eebb1bfb7c7e8b240a6a3cb3ab24479ea294f3539429574b1eec862
Mac	commandlinetools-mac-11076708_latest.zip	153.6 MB	7bc5c72ba0275c80a8f19684fb92793b83a6b5c94d4d179fc5988930282d7e64
Linux	commandlinetools-linux-11076708_latest.zip	153.6 MB	2d2d50857e4eb553af5a6dc3ad507a17adf43d115264b1afc116f95c92e5e258

Command-line tools are included in Android Studio. If you do not need Android Studio, you can download the basic Android command-line tools above. You can use the included `sdkmanager` to download other SDK packages.

● Use the following instructions to set up Android command lines correctly in your WSL2.
    Open WSL2 terminal.
    Create a new directory in your home directory.

~ $ mkdir myAppName
~ $ cd myAppName

```
$ ~ $ mkdir Android
~ $ cd Android
```

Unzip the Android command line tools zip file in the android directory using this command: unzip ~commandlinetools-linux-11076708_latest.zip

● In the Android directory perform following actions.

```
$ mkdir myAppName
$ mv cmdline-tools/* myAppName/
$ mv myAppName/ cmdline-tools/
```

● Now add the following line to your ~/.bashrc file

```
export ANDROID_HOME=$HOME/Android
export PATH=$PATH:$ANDROID_HOME/cmdline-
tools/latest/bin/:$PATH
```

● Save your ~/.bashrc file and exit by typing qw;

● Run source ~/.bashrc to reload the bashrc file.

● Now, we will install basic Android utilities using the following
commands:
Run sdkmanager --list to check if everything is working fine.
Run sdkmanager --install "platform-tools" to install platform tools.

● Finally, add the following into your ~/.bashrc file

```
export PATH=$PATH:$ANDROID_HOME/platform-tools/:$PATH
Save your ~/.bashrc file and exit.
Run source ~/.bashrc to reload the bashrc file.
```

● To check that everything went well, do the following:
Close and relaunch terminal
Run adb --version and see that adb version is shown
Since everything is installed fresh, your WSL 2 adb version should
perfectly match with Windows ADB version that we noted down
as part of the pre-requisites.

**Please follow the below steps to setup the ADB and make sure you are
able to use Android emulators with your WSL2 correctly**

● Fire up your Android emulator on Windows.
● Once the Android emulator is up and running, open a PowerShell
prompt.
● Run this command in PowerShell

```
adb -a -P 5037 nodaemon server
```

> This will start the adb server in the Windows host.
> Note down the IPV4 address of your Windows host
> PC/machine.

Move to the directory of your project (myApp)

● Install Maestro :

Installing Maestro is now just a matter of running following one
command.
```
curl -Ls "https://get.maestro.mobile.dev" | bash
```
```
$ curl -Ls "https://get.maestro.mobile.dev" | bash
```

● Check your Maestro version using the following command:
```
maestro --version
```

● **Note: Don't close the PowerShell terminal!**
● **Now open your WSL2 terminal and run these commands.**
```
adb kill-server
export ADB_SERVER_SOCKET=tcp:<WINDOWS_IPV4_ADDR>:5037
adb devices
```
You should see your connected emulator successfully now.

Step 5: To write your first Maestro code :

● You can refer to the following documentation
https://maestro.mobile.dev/getting-started/run-a-sample-flow

● To Start writing a Simple code referring to the above document you
can download the sample code using the following command in your
Ubuntu terminal:

```
Maestro download-samples
```

● **To run your sample application run the associated flow using the
Maestro test command :**

```
cd ./samples
```

```
adb install sample.apk
maestro test android-flow.yaml
```

● You can write this code for creating your own application

```
#flow: Login
#intent:
# Open up our app and use the default credentials to login
# and navigate to the demo screen

appId: com.midas # the app id of the app we want to test
# You can find the appId of an Ignite app in the app.json file
# as the "package" under the "android" section and "bundleIdentifier" under the
"ios" section
---
- clearState # clears the state of our app (navigation and authentication)
- launchApp # launches the app
- assertVisible: "Sign In"
- tapOn:
    text: "Tap to sign in!"
- assertVisible: "Your app, almost ready for launch!"
- tapOn:
    text: "Let's go!"
- assertVisible: "Components to jump start your project!"
```

```
#flow: Login
#intent:
# Open up our app and use the default credentials to login
# and navigate to the demo screen

appId: com.midas # the app id of the app we want to test
# You can find the appId of an Ignite app in the `app.json` file
# as the "package" under the "android" section and "bundleIdentifier" under the "ios" section
---
- clearState # clears the state of our app (navigation and authentication)
- launchApp # launches the app
- assertVisible: "Sign In"
- tapOn:
    text: "Tap to sign in!"
- assertVisible: "Your app, almost ready for launch!"
- tapOn:
    text: "Let's go!"
- assertVisible: "Components to jump start your project!"
```

● If you want to use the sample code which opens the Wikipedia
  through Maestro you can use the following command:
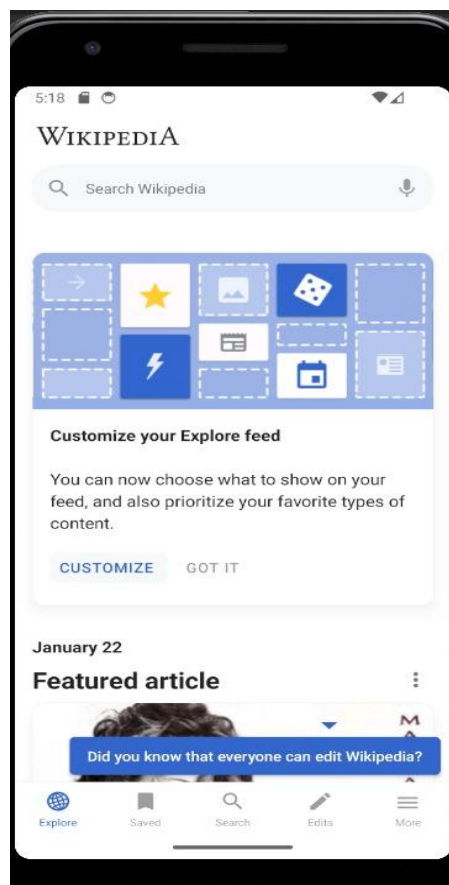
```
appId: org.wikipedia
---
- launchApp
```
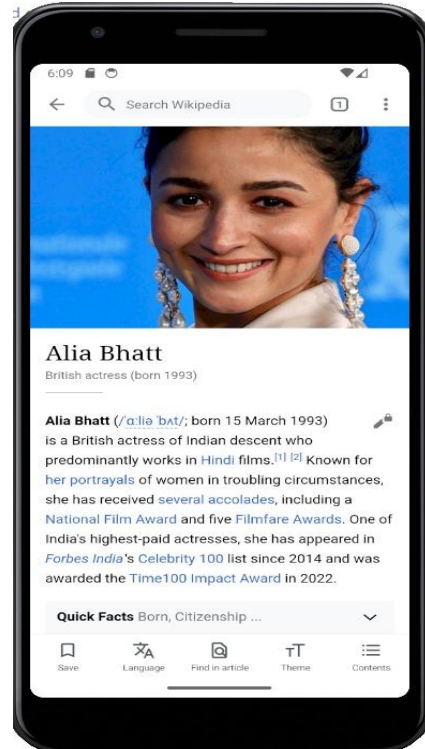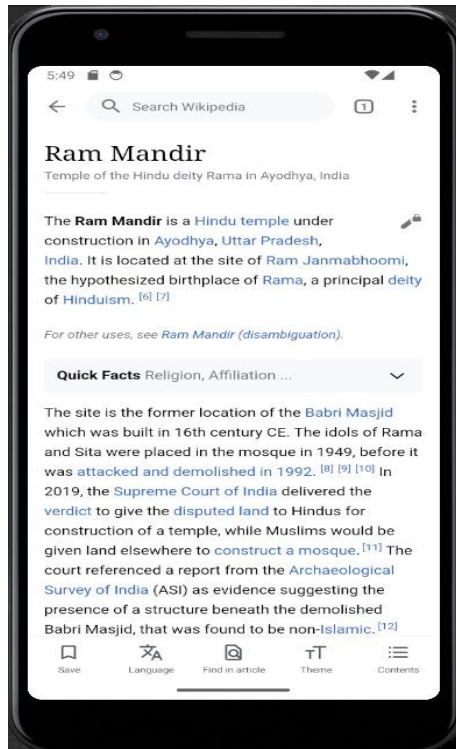
Run the following command in WSL2 :

```
$ maestro --host <your ip address> test android -filename.yaml
```

The result shown in the WSL Terminal should be :

```
> Flow
    ✅ Launch app "org.wikipedia"
```

Result Shown in Emulator :

Name: Yash Anand
Roll no: 20051769