INTERNATIONAL UNIVERSIT*Y* - VNU - HCMC

Special Study of The Field

***Report***

**Ngu*y*ễn Thành Thiện – ITITIU14089**

**I. ABOUT REACT NATIVE**

- React Native is an open source provided by Facebook which is a framework that lets you build native iOS and Android apps with one unique language of Javascript. React Native provides a virtual DOM (Document Object Model) which is a generic representation of what your UI will look like independent of the two platforms. React Native then renders the native control based on your UI markup representation which then renders native controls. It is able to take native platform components (sliders, switches, labels, tab bars) and wrap them in React component counterparts.

- Goal: “be able to develop a consistent set of goals and technologies that let us build applications using the same set of principles across whatever platform we want.”

**-** Language: JavaScript syntax, especially is ECMAScript2015 (ES6) - the new version of JavaScript which has a lot of new JavaScript language features like class syntax, destructuring, properties, etc.

**-** React is still new but it’s maturing quickly and Facebook has stated it plans to continue investing in its growth.

**II. WHY REACT NATIVE?**

**1. Saving time and reducing development costs.**

***a. Easy to learn***

- Getting started with React Native is easy - especially for JavaScript professionals. Just download the open-source code from Github and make sure you’re familiar with a few tools and constructs in the React Native library including NodeJS, the CSS Flexbox system, ECMAScript 6, and JSX.

### *b. Cross-platform*

- React Native was originally only developed for iOS support, but due to its success and popularity, Facebook decided to develop support for Android as well, so which translates to lower development costs.

***c. The ability to combine good features and build apps***

- Whereas native app development is usually associated with inefficiency, less developer productivity, and slower time to deployment, React Native brings the speed and agility of web app development to the hybrid space with native result (the combination between pros of Hybrid and Native apps).

- React Native provided "Hot Reloading" (the previous version is “Live Reloading”) that keeps the app running and to inject new versions of the files that you edited at runtime. By this, you don't lose any of your state which is especially useful if you are tweaking the UI. The result is times between you save a file and be able to get this feedback loop can be under 1 second, even as your app grows.

**2. Performance**

- Traditionally, “Hybrid” or “Mobile web” apps that work on both iOS and Android tends to go down for more complicated apps. Apps built through React Native are compiled into natively written code, so they not only work on both operating systems, but also function the same as a natively written app (**“learn once, write anywhere”**).

- React Native is focused solely on building a mobile UI, making it more like a JavaScript library than a framework. It is developed based on Facebook’s popular ReactJS UI library for web applications, so it brings all of ReactJS’s better app performance: DOM abstraction, and simplified programming methods to hybrid mobile development. The resulting UI is highly responsive and feels fluid (the app will have quicker load times than a typical hybrid app, and a smoother feel) thanks to asynchronous between Javascript and Native.

**3. Reusability**

- Reusability is key in React Native. That mean the building blocks are treated as native components that can be compiled directly into the native languages (Objective-C for iOS and Java for Android). This is a huge bonus for businesses that want to augment an existing app but don’t want to overhaul it by Incorporate React Native components into the app’s code.

- This is possible to exclude the WebView components of other Hybrid mobile apps because React Native’s building blocks are reusable “native components” that compile directly to native. For example, if you’re adding Google Maps functionality to your app, React Native lets you link the plugin with a native module without have to rely on a WebView for certain functions. Components you would use in platforms have counterparts right in React, so you’ll get a consistent look and feel.

**4. Maintainablity**

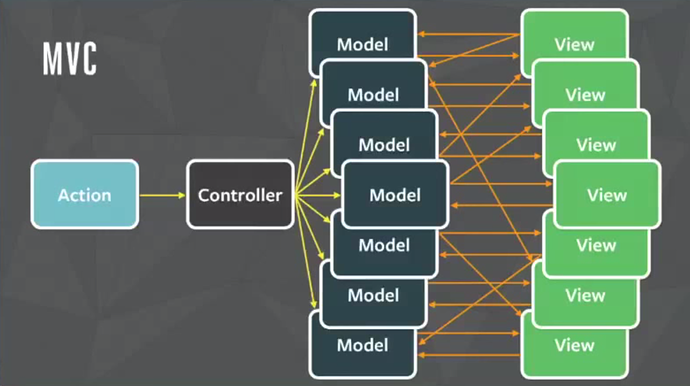
- React Native uses Javascript and this, coupled with the intuitive architecture of the framework itself, allows engineers to jump to and from each other’s projects on Native apps with relative ease.

- The typical workflow would consist of writing most of the code on React Native and then using the native languages to optimize certain elements of the app. Parts of a specific app written in React Native will have no trouble combining with parts written in the native languages for iOS or Android. It is completely up to the developer to build the app, it not only can the native languages of an OS be used to optimize an app, they can also be used to write parts of the app.

**III. THE ARCHITECTURE OF REACT**

- React is only the View layer (V) in the MVC - Model - View - Controller architecture. All the while you see that React is mentioned along with other frameworks, but it offers just the View, It has no M (Model) or C (Controller).

- One of the easiest ways is combine React with other MVC frameworks, such as Backbone, or even Angular, but when the codebase and code organization are expanded, the MVC model "*MVC got really complicated really quickly*". When your application grows and grows, it will be difficult to debug and control the code if it is not well organized => “MVC DOESN’T SCALE!”

*Figure 1.1 - The uncontrollable of MVC to Facebook’s huge codebase (Source: Internet)*

- A solution for the problem of using React effectively, Facebook proposed a new architecture, called **Flux**. According to Facebook engineers, Flux will make their massive code more understandable, easier to debug, and especially "**predictable**”.

**IV. WHAT IS FLUX?**

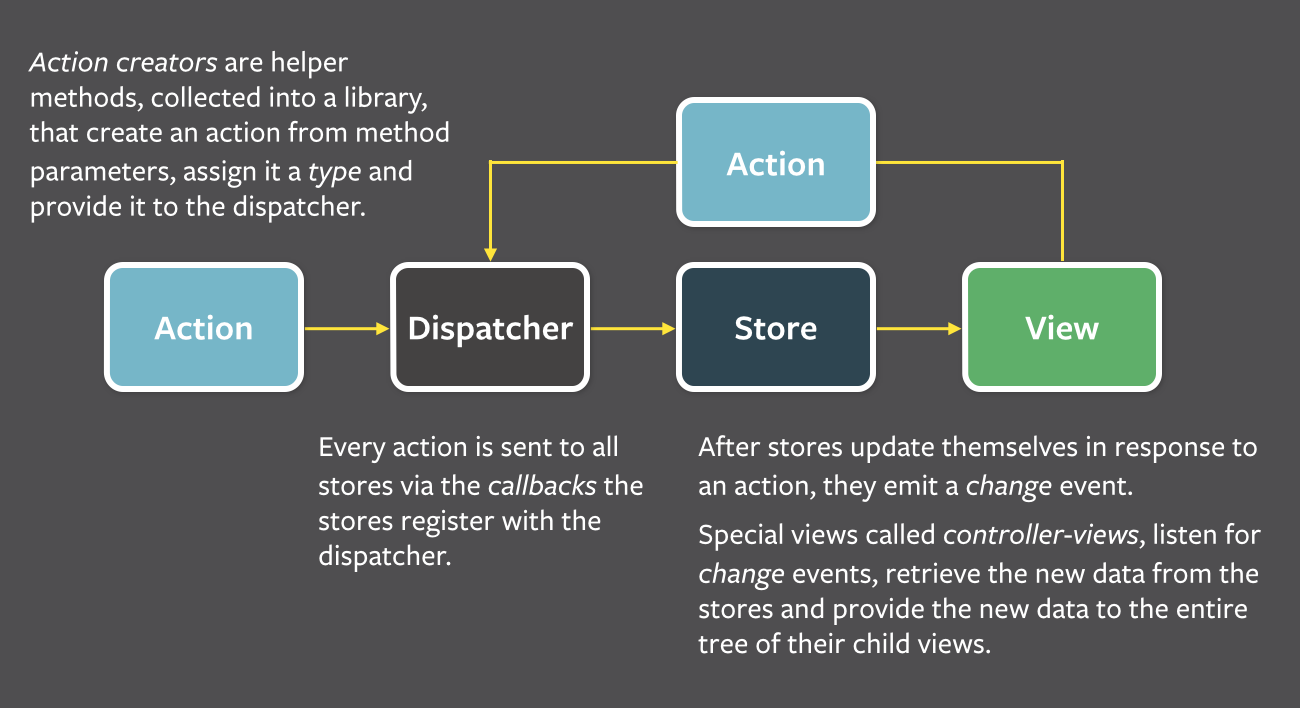
- Flux is an architecture for creating data layers in JavaScript applications. It was designed along with the React view library. It places a focus on creating explicit and understandable update paths for your application's data, which makes tracing changes during development simpler and makes bugs easier to track down and fix.

- Flux consists of the following basic components:

**+ Action:** This is the place to register the functions that will be called when View needed.

**+ Dispatcher:** Has the role of transmitting calls from Action to the Store. When an Action is called, the Dispatcher will broadcast an event to all Stores, with Action Type, or any other required data.

**+ Store:** The place to store data, is also the only place where you can add, edit or delete data. The store will listen to the events that come from the Action through the Dispatcher, check that the event is under its processing, and make the necessary data changes, corresponding to each event. After changing the data, Store will give another event to report its change.

**+ View:** Retrieves data from functions.

*Figure 2.1 - Flux architecture (Source: Internet)*

- React using **one-way data bindings**, because they think that two-way data bindings will become "When one object update, it many other objects will update, and then makes more updates". With one-way, when all the changes go to the Dispatcher, they can easily find where the change comes from, so the system will become "predictable".

- Flux follows the concept of **Unidirectional Data Flow** (UDF is to keep the data flow in the application moving in a single direction. When the data changes, this stream reboot from scratch) making it much easier to zero in of where the error lies. The data goes through a strict pipeline through your application:

+ Any changes, or Actions, must go through the Dispatcher!

+ Store is just only public getter, not public setter. Data change handlers can only be called inside the Store itself. This means:

+ You can not change the data in the Store directly from the View.

+ You also can not change the data in one Store from another. As mentioned above, Store is not public setter. To do that, you must go through Action and Dispatcher. This makes the data inside your Store more manageable, you can easily debug where the change comes from when data changes.

**V. THE DIFFERENCES OF FLUX (WITH MVC)**

**1. Easier to understand the data flow.**

- In the bidirectional data flow, you have the typical data flow   MVC. But when applications became more complex, the Controller takes the huge responsibility of maintaining both the application state and the data. Also, the cascading updates makes the app really difficult to understand and debug.

- With UDF, changes in the application view layer will trigger an action in the data layer. These changes will then be reflected in the View. The View does not directly affect application data.

**2. Reducing the role of Controller.**

- Dispatcher is not a controller, it does not contain business logic. It simply is a "coordinating center", which send an Action to every Store. In the MVC model, you can design any Controller as you can. But in Flux, there is only one Dispatcher and all Actions must go through it. The appearance of the Dispatcher is important because it ensures the design of UDF.

**3. Store can handle many objects.**

- In the MVC model, the Model usually manage a particular object. In opposition to the Model, the Store may not handle any data or handle multiple states of the Application, or manage multiple objects at once.

**VI. IMPLEMENTATIONS OF FLUX**

- **Flux is an Architecture**, not a Framework. Facebook presents the ideas of that architecture, and you can create a framework or library of your own Flux architecture. There are some commonly Javascript libraries that help to write Flux app:

**1. Redux**

- According to Github, Redux is a **predictable state container** for JavaScript apps. Many of the concepts are similar to functional programming, and all the data is kept in a single store (the state is immutable).

- The updates on the application state is done through Actions, which are plain objects themselves, but contains a type property depicting the kind of action performed.

**2. Reflux**

- It is one of the most popular implementations of Flux. But there are some basic differences between the two. Reflux doesn’t use a Dispatcher; instead each Action is a Dispatcher.

- As the Actions themselves are functions, there are no action creators. And the best thing about Reflux is that it is more concise and streamlined, with absolutely very less requirement of repetitive codes.

**3. Fluxxor**

- Fluxxor makes use of a number of tools and Flux architecture to build JS data layers. In order to enjoy the full functionality of Fluxxor, you will have to make it work with React as the view layer.

**4. Alt**

- Modeled after Flux, Alt is a library that facilitates the managing of state in JavaScript applications. You can install Alt, if you are installing package manager like nom or bower. Alt gives you the benefit of Flux, but with a better syntax.

**VII. SUPPORTING TOOLS**

**1. Environments work with React Native**

- NodeJS: When you start actually building something, you’ll want to have NodeJS. Download NodeJS at: <https://nodejs.org/en/>

- npm: To help install packages and dependencies for Javascript developers. npm is distributed with Node.js- which means that when you download Node.js, you automatically get npm installed on your computer. You can also create an app by using Create React Native App (CRNA) with npm:

npm install –g create-react-native-app

create-react-native-app *your-project-name*

cd *your-project-name*

npm start

**2. Expo: Supporting build tool**

*a. What is Expo?*

- Expo is a framework base on React Native and other React Native libraries to create a build tool of application quickly. It helps developers deploying and developing their React Native app for both iOS and Android platforms easily. Here are some special features of Expo:

+ Expo is free, it might be charged money for services built on top of Expo or for some kind of premium level of support and consulting.

+ Completely using Javascript.

+ No need to install any other build tools (XCode - Apple, Android Studio - Google,...).

+ Expo is open source, so it can add other quality and variety libraries from the community, gain needs of most applications.

+ Updating source code is not required through censorship of the Apple Store or Google Play, just only static file updates such as Javascript, CSS, Image ...

+ Expo will build your app and output a link to the binary required for you to submit. Then you can use something like Application Loader for iOS, or directly upload an APK for Android.

+ Push Notification, Facebook Login is easier.

- Cons of Expo:

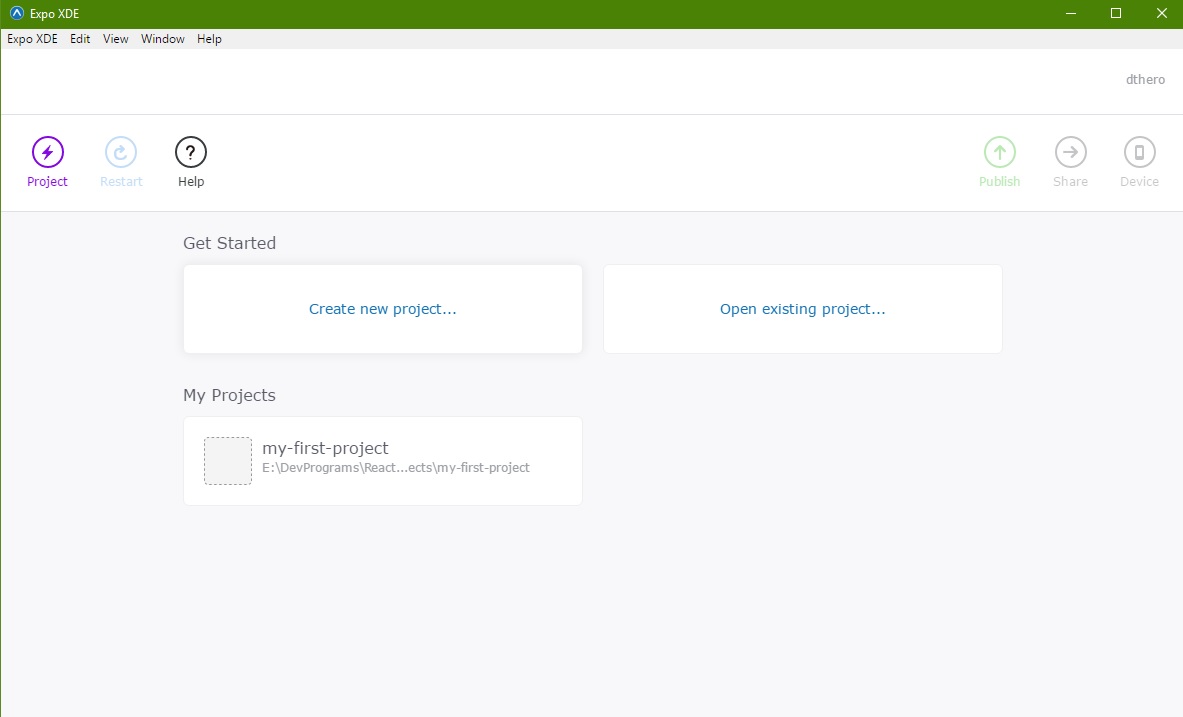
+ Standard Expo projects don’t support custom native code, including third-party libraries which require custom native components. In an Expo project, you only write pure JS. But it is possible to detach and use the Expo's libraries through ExpoKit, it means that what you wrote on Expo will work normally and you may use other native code libraries to continue.

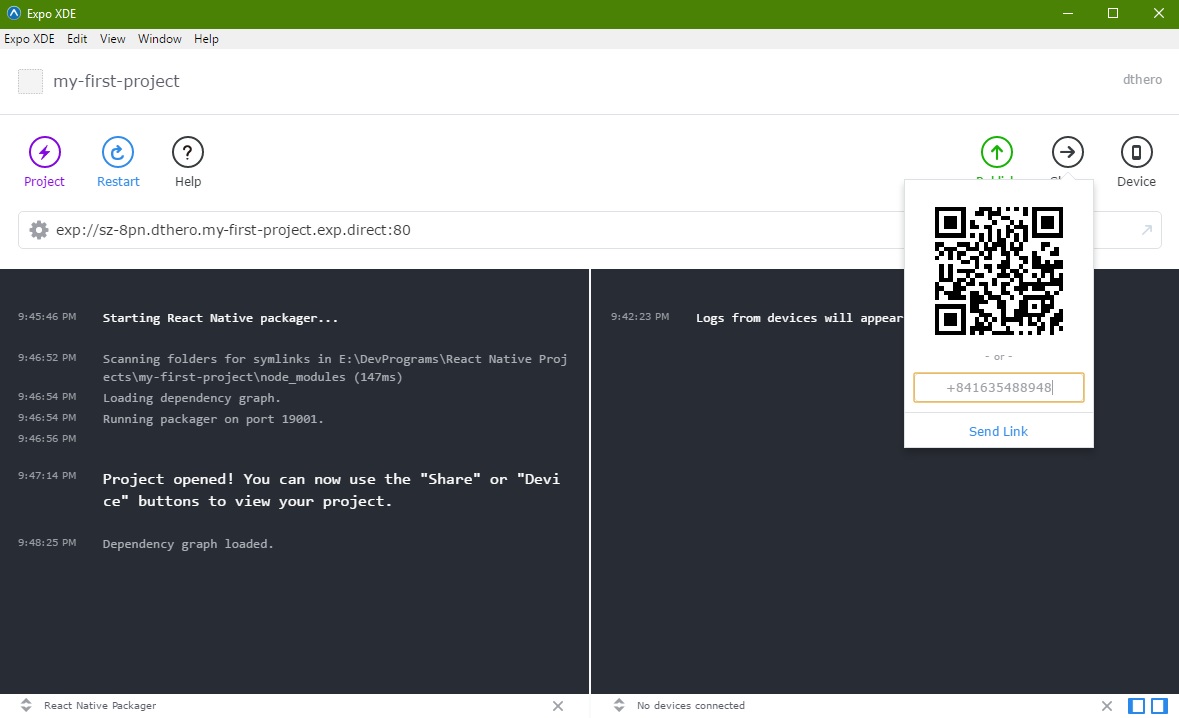
+ Expo is a new build tool, so it is incomplete and in developing mode (the most obvious is hard to create new React Native project by Expo XDE on Windows).

*b. Using Expo*

- You can use Expo XDE (a development environment available on macOS, Linux and Windows) or CLI by Exp (terminal program) to create and build a mobile app:

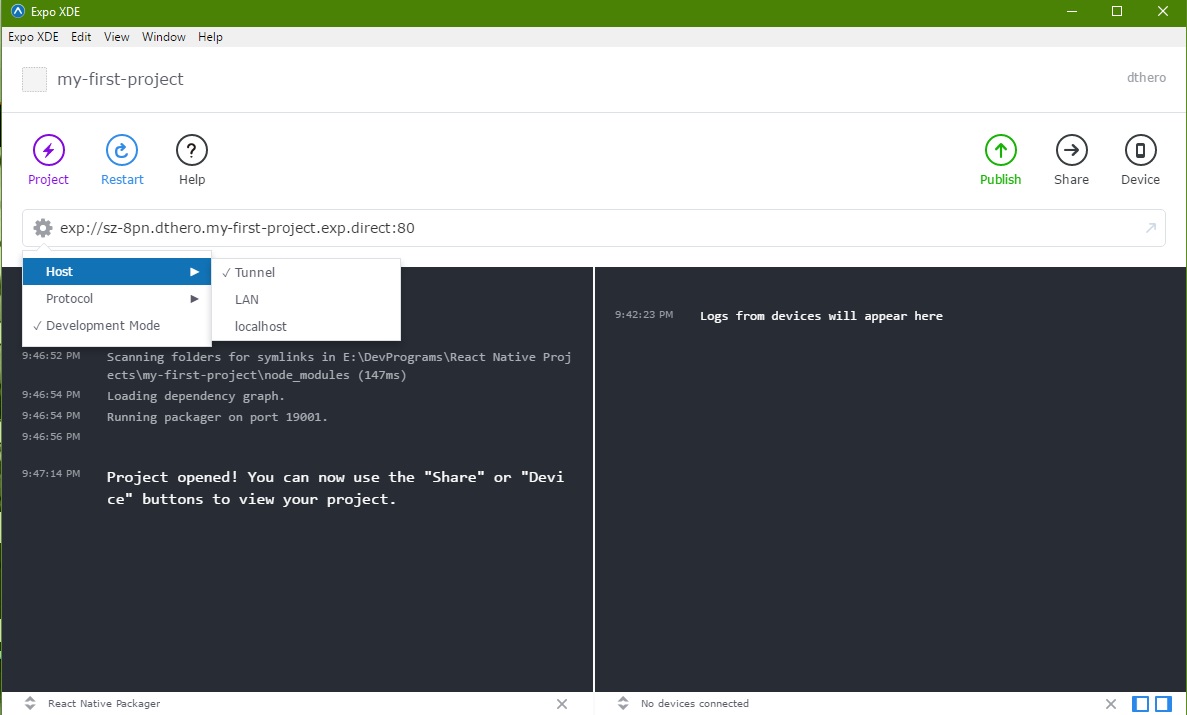
npm install exp --global  
exp init *your-project-name*  
cd *your-project-name*  
exp start

*Figure 3.1 - Expo XDE main screen*

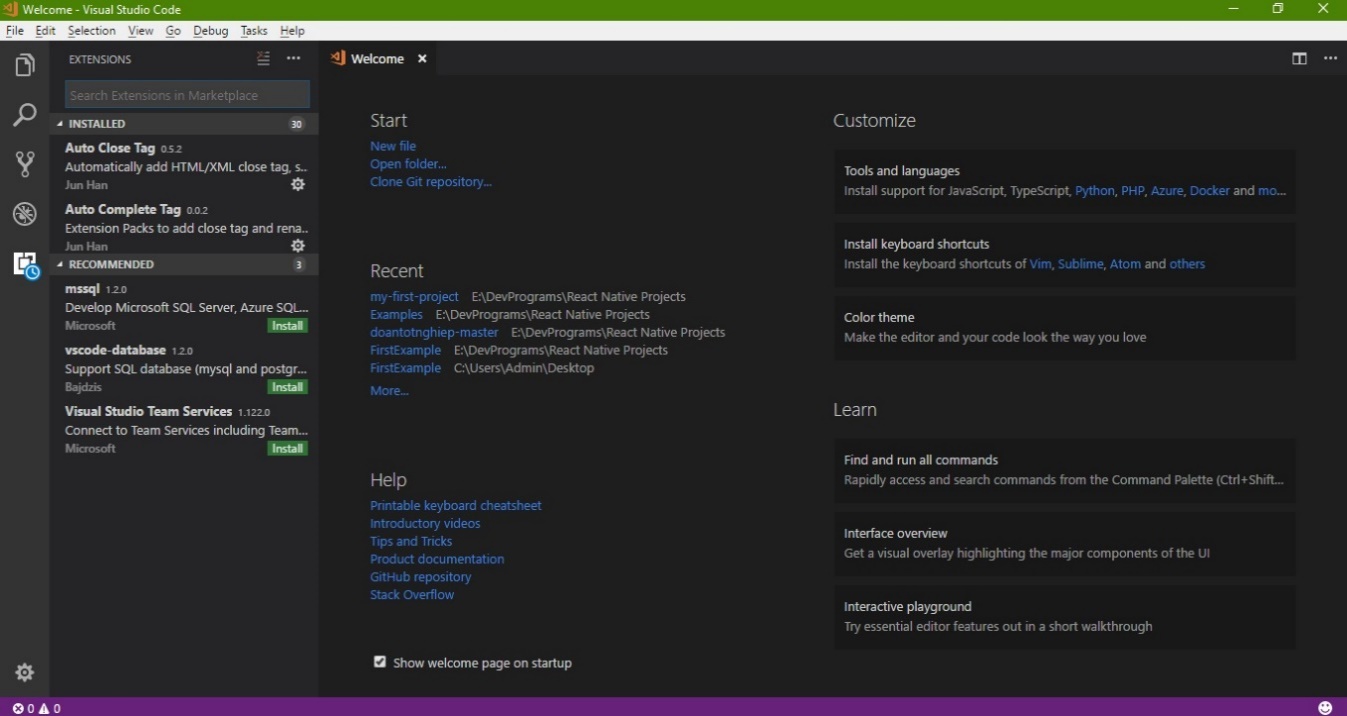
- Expo using QR code (or Phone number) to share app (Example using CLI: exp send *your-share-phone-nnumber*). You just need to install the Expo mobile app (on App Store or CH Play) into your specific device to preview projects and test code live.

*Figure 3.2 – Share/Send project to specific device by Expo XDE*

- You can Publish app by clicking ‘Publish’ in XDE or running exp bulish in your project. This gives your app a URL.

*Figure 3.3 - Configure private network on Expo XDE*

- For more information, please refer: <https://docs.expo.io/>

**3. Visual Studio Code: Implementation tool**

*Figure 3.4 – VS Code interface*

- Because I work with Expo on Windows and Linux, so I use Visual Studio Code (VS Code) to implement and fix code (you can use XCode on macOS or Sublime Text, Atom,... instead).

- Find and download VS Code at: <https://code.visualstudio.com/>

*a. Pros of VS Code:*

- Simple UI, there are some sidebar, which can be left or right to your liking.

- In this panel you can search anything in Command Palette (Ctrl + P) such as quick and easy actions, Git, change theme, install extension,...

- Git integration. The changes you make in your files are marked different so you can know exactly what is going on when you commit, push,...

- Split screen - side by side viewing, be used to compare the file easily.

- Easily to find and install Extensions (Intellisense, Auto Complete, Beautiful Code, ESLint,...).

- Quick fix and supporting debug very strong.

- Implementing code quickly by using key binding, to see the common Keyboard Shortcuts table press: Ctrl + K + S.

*b. Suggestion about extensions for simply implementing React Native in VS Code*

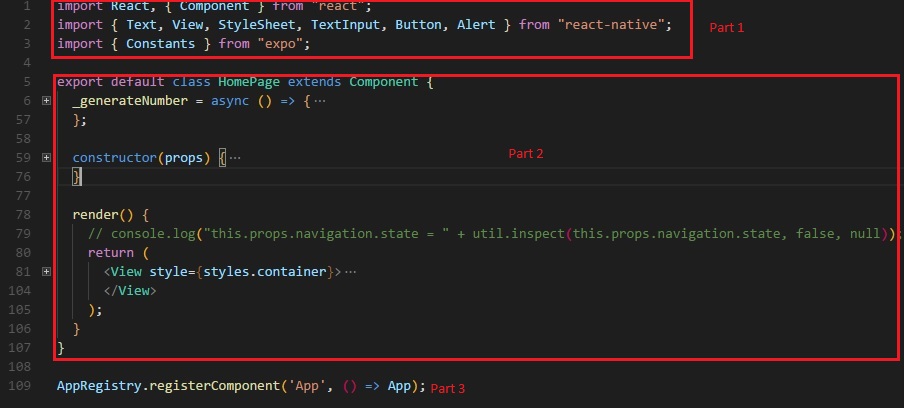
- React Native Tools.

- Flow Language Support: To catch errors and display it either in our editor or when we run Flow from the command line.

- ESLint: To provide a pluggable linting utility for JavaScript.

- Prettier - JavaScript formatter.

- Path Intellisense, Auto Complete Tag, Git Blame,...

**4. Basic project React Native**

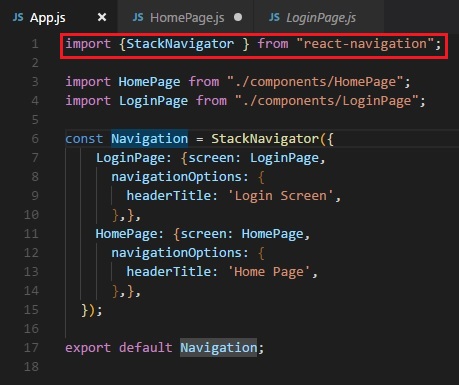
*Figure 4.1 – Structure of a file*

- A basic file in React Native have 3 main part:

***+ Import:*** To import components/libraries that the file needs to use.

***+ Component:*** Like a class, you must have a class extends from component to handle your code. Inside component is *Render()* to return what you want to show in the screen

***+ AppRegistry:*** To define that what component will be default (using first) in your app.

- Navigator is used to navigate from one component to another.

*Figure 4.2 – Using Navigator*

*Figure 4.3 – View in screen*

*Figure 4.4 – Style in React Native*

- Style (like the CSS fle in Web) is used to describe what elements look like.

- Code publish using Expo or publish on Git.

**5. Some open source code used to refer**

- **Snowflake:** <https://github.com/bartonhammond/snowflake>

- **Tinder:** <https://github.com/VctrySam/Tinder>

**- React-Native-NBA-App:** <https://github.com/wwayne/react-native-nba-app>

**- React-Native-NW-React-Calculator:** <https://github.com/benoitvallon/react-native-nw-react-calculator>

**- Finance React Native:** <https://github.com/7kfpun/FinanceReactNative>

\* In all projects that I read, most of them followed Redux for designing pattern and structure. Thus, I will focus on **Redux**.

**VIII. ABOUT REDUX:**

- Redux is a library with minimal API but completely predictable behavior, so it is possible to implement logging, hot reloading, time travel, universal apps, record and replay, without any buy-in from the developer.

- Redux can be used together with React, or with any other view library, and following 3 principles:

+ All application state is contained within a ***single store***, which is most often a JavaScript object.

+ The application's state is ***immutable***. This means that at no point should the object representing the state be modified in any way by any component.

+ All functions that compute the new state (the so called reducer functions) must be **pure functions**. Pure functions are functions that produce no side-effects and are deterministic - for a given set of inputs, the output will always be the same.

- General structure of Redux design pattern:

+ ***actions*:** Are payloads of information that send data from your application to your store. They are the only source of information for the store

+ ***reducers*:** Reducers specify how the application's state changes in response.

+ ***stores*:** The actions that represent the facts about “what happened” and the reducers that update the state according to those actions. The Store is the object that brings them together.

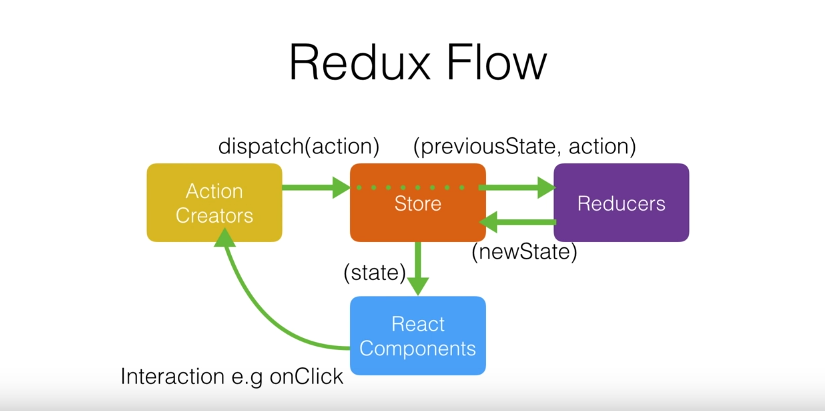
+ ***components:*** Folder contains the components that is only responsible for view and receive user interaction (How things look: markup, styles). These files read and change data from *props* => Dump components.

+ ***container:*** The components that are aware of Redux, Router, etc. They are more coupled to the app. They transmit the data to dump components with *props* (How things work: data fetching, state update) => Smart components.

- Other kinds of data in structure:

+ ***A main Javascript file:*** Main component of app, which decides the first component will be called.

+ ***images:*** The folder that contains all the image sizes and formats for app.

+ ***styles:*** If you want to simple your components, put your css styles in specific folder and then call it in component by import.

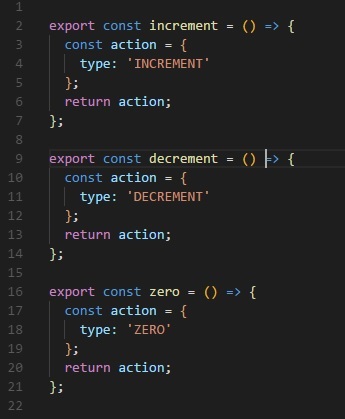
*Figure 5.1 – Redux flow (Source: Internet)*

**\* Implement basic Redux:**



*Figure 5.2 - Store*

- Redux has only 1 store using createStore() function to initialize store and has reducer function (have 2 params are current “state” and “ACTION”) to handle action. Reducer will return new object instead of return state (since “state” is immutable).



*Figure 5.3 – Actions*

****- Actions provide payload data and a constant of string to define an action.

*Figure 5.4 – Dispatch in Redux*

- In Redux, Dispatcher is removed. Store will call dispatch function directly.

****

*Figure 5.5 – State in component*

- When the app’s state change, we use getState() function to fetch app’s state into component, and using subscribe() function to listen the change.