

Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

Asignatura: Mini Proyecto de Desarrollo de Software (EP4793)

**Desarrollo de la aplicación móvil LithoDex**

**(Fase 3)**

**Documento técnico**

Docente: Mireya Morales Alumno: Ronald Becerra

Carné: 12-10706

2020-2021

**Tabla de contenidos**

* Información general
* Tecnologías
* Instalación del ambiente de desarrollo
* Obtención del proyecto de la fase 3
* Instalación de nuevas bibliotecas
* Emular la aplicación
* Mensajes de advertencia y errores al emular
* Compilar la aplicación
* Añadir permisos
* Creación de un nuevo proyecto React Native
* Expulsar Expo del proyecto
* Enlaces y referencias

**Información general**

LithoDex es un sistema informático que facilita el levantamiento de información en trabajos de campo del área de geología. Su finalidad es brindar una herramienta que les permita a los geólogos almacenar directamente de forma digital los datos que recolectan en sus trabajos de campo, ahorrando el trabajo adicional de hacer la recopilación en papel para luego pasarla manualmente a una computadora. Por ello, LithoDex fue pensado en primer lugar como una aplicación destinada a usarse en dispositivos móviles, que son los que se pueden trasladar con facilidad a los lugares de estudio, y dicha aplicación es la que se ha comenzado a desarrollar por el momento. También se ha planteado extender el sistema de modo que tenga una vista adicional en PC’s, en la que se puedan manipular con más facilidad los textos, los dibujos y las fotografías que se obtuvieron con los dispositivos móviles, pero eso no es un objetivo primordial y no se ha comenzado a trabajar en ello.

Este documento tiene como objetivo mencionar qué herramientas tecnológicas han sido empleadas en el desarrollo de la aplicación y explicar cómo un nuevo programador podría hacer las configuraciones necesarias para continuar trabajando en la misma, además de señalar algunas consideraciones técnicas que deben tomarse en cuenta.

**Tecnologías**

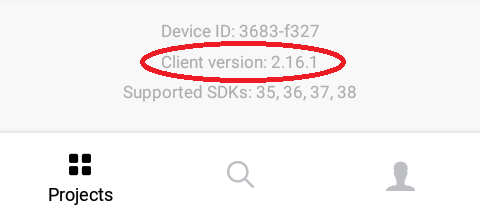
En la computadora en la que se realizó la programación se utilizó mayormente el sistema de operación Windows 10 de 64 bits, aunque para las compilaciones se empleó también Ubuntu 14.04.6. En los dispositivos móviles en los que se ejecutó la aplicación se utilizó el sistema de operación Android, de versiones 5.1.1, 6.0 y 9.

Se empleó el marco de trabajo (*framework*, en inglés) **React Native**, que requiere programar en el lenguaje JavaScript, y que luego permite transformar el producto a código nativo de múltiples sistemas de operación de dispositivos móviles. Gracias a esto la aplicación debería poder compilarse tanto para Android como para iOS, que son los sistemas móviles más utilizados actualmente. Trabajar en React Native hizo necesario a su vez el uso de **Node.js**, que es un entorno de ejecución de JavaScript que en este caso sirvió como un conjunto de herramientas de desarrollo auxiliares para el proyecto. Por otra parte, trabajar con React Native puede hacerse de forma “pura” o a través de una herramienta auxiliar llamada **Expo**, que facilita el desarrollo de aplicaciones móviles de Android y iOS con este marco de trabajo. La decisión sobre si usar Expo o no debe tomarse al momento de crear un proyecto nuevo. En este caso lo que se hizo fue continuar trabajando en un proyecto que ya se había comenzado con Expo en fases anteriores. Algunas de las facilidades que provee Expo son:

1. Emular la aplicación en un móvil sin necesidad de compilarla, manteniendo los códigos fuente en un servidor (computadora). Esto es útil mientras el proyecto se está desarrollando, ya que así podemos ver los cambios reflejados rápidamente sin necesidad de hacer una nueva compilación.
2. Permite construir con comandos sencillos la versión compilada de la aplicación, que es la que se puede instalar permanentemente en un dispositivo móvil. Expo tiene además la ventaja de que si se hace una nueva compilación de la misma aplicación, los dispositivos que ya la tenían instalada recibirán automáticamente las actualizaciones, por lo que no tendrán que instalarla de nuevo. Si el programador deseara evitar que recibieran tales actualizaciones, podría cambiar la versión de la aplicación antes de compilarla, la cual está indicada en un archivo llamado “app.json”, ubicado en el directorio raíz del proyecto.

Ahora indicaremos las versiones de las tecnologías. Debe tomarse en cuenta que ésas son las últimas que se emplearon en este desarrollo; no necesariamente son las mismas que se utilizaron desde el comienzo porque en más de una ocasión fue necesario actualizarlas. También explicaremos cómo se pueden consultar dichas versiones. Cuando una ruta mencionada comience con “./”, estaremos asumiendo que ya se está ubicado en el directorio raíz del proyecto “lithodex”, que debe haber sido descargado por el nuevo programador (véase la sección **Obtención del proyecto de la fase 3**).

* React Native 0.62.2 🡪 Esto puede consultarse en el archivo “./node\_modules/react-native/package.json”, en donde dice “version”.
* React 16.8.3 🡪 Esto puede consultarse en el archivo “./package.json”, en la sección “dependencies”, en donde dice “react”.
* Expo 4.0.12 🡪 Es la versión de Expo que se usó en el servidor de LithoDex, es decir, en la computadora. Se puede verificar usando la terminal y escribiendo “expo -V” o “expo-cli -V”. Nótese que la “V” debe estar en mayúscula. También se puede colocar “expo -Version”, o bien “expo-cli -Version”. Además, no es necesario estar ubicados en la ruta del proyecto porque esta herramienta se instala para uso global en la computadora, no para un único directorio específico.
* Expo SDK 38.0.0 🡪 La herramienta Expo SDK provee acceso al dispositivo móvil y a varias de las funcionalidades de su sistema de operación, tales como contactos, cámara y ubicación GPS. Las siglas SDK son el acrónimo en inglés de *Software Development Kit* (Equipo de Desarrollo de *Software*). Se instala automáticamente por defecto en cualquier proyecto Expo, por lo que el usuario no tiene que hacer su instalación aparte manualmente. La versión del SDK puede verificarse en el archivo “./app.json”, en la parte que dice “sdkVersion”.
* Expo: Client Version 2.16.1 🡪 Es la versión de Expo para dispositivos móviles que se instaló en los teléfonos en los que se hicieron pruebas de la aplicación. Esta versión soporta los SDK’s 35, 36, 37 y 38, por lo que fue compatible con la que se estaba usando en el servidor, que es la 38.0.0. Para verificar la versión en el dispositivo móvil, basta con ingresar en la aplicación Expo y revisar en la parte inferior de la vista (Figura 1).



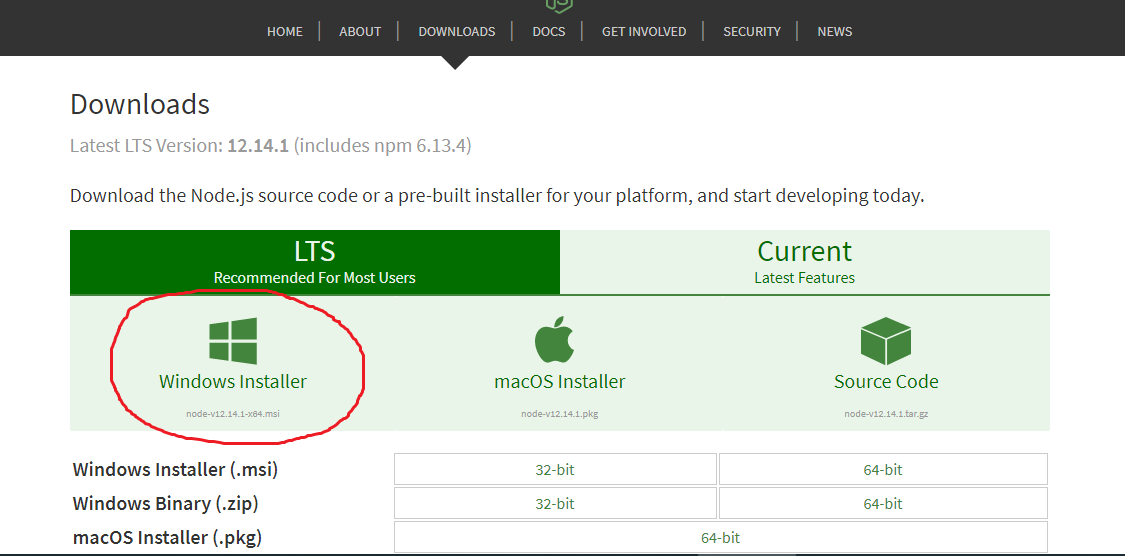
**Figura 1**. Versión de “Expo: Client Version”.

* Node.js 12.16.1 🡪 Se puede verificar usando la terminal y escribiendo “node -v”. Nótese que la “v” debe estar en minúscula. Además, no es necesario estar ubicados en la ruta del proyecto.
* NPM 6.13.7 🡪 Es el instalador de paquetes por defecto de Node.js, por lo que viene automáticamente integrado con él. La versión se puede verificar usando la terminal y escribiendo “npm -version”, o bien “npm -v”. Nótese que la “v” debe estar en minúscula. Tampoco es necesario estar ubicados en la ruta del proyecto.
* Yarn 1.21.1 🡪 Es una herramienta para instalar paquetes similar a NPM, pero que tiene la ventaja de ser más rápida. Hay que instalarla por separado ya que no viene por defecto con Node.js. La versión se puede verificar usando la terminal y escribiendo “yarn -V”. Nótese que la “V” debe estar en mayúscula. Tampoco es necesario estar ubicados en la ruta del proyecto.

**Instalación del ambiente de desarrollo**

1. **Lo que necesitamos tener en nuestro computador**

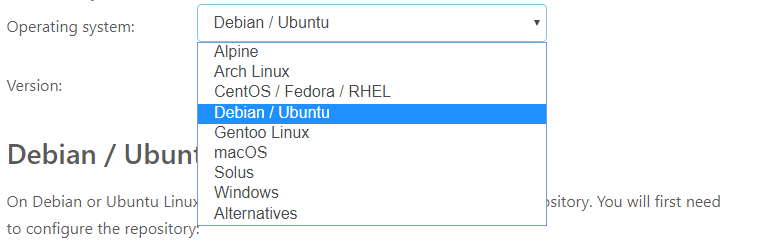
* **Node.js**. Para obtener la última versión vamos a su sitio oficial para descargas en [1]. En la Figura 2 puede verse la pestaña “LTS (Recommended For Most Users)”, y justo debajo unas tres opciones: “Windows Installer”, “macOSInstaller” y “Source Code”, aunque el orden en que aparecen puede variar dependiendo del sistema de operación que estemos utilizando. Si, por ejemplo, estamos en Windows, seleccionamos la opción “Windows Installer”, que se ve resaltada en la imagen, y eso descarga el instalador. Luego lo abrimos y seguimos las instrucciones para ejecutarlo.



**Figura 2**. Selección del instalador de Node.js según el sistema de operación.

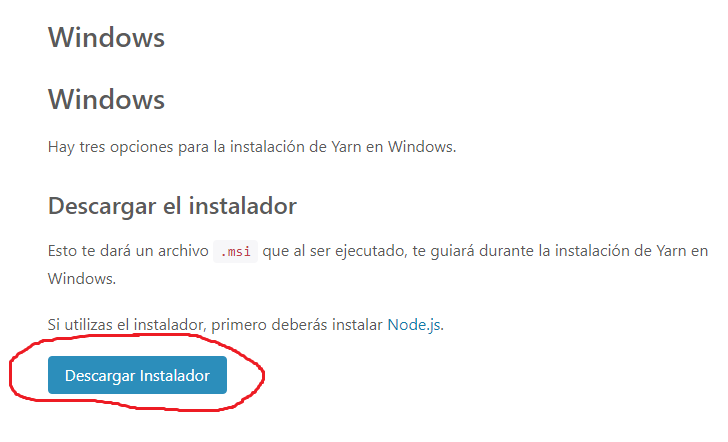
* **Yarn**. Esto permitirá instalar a su vez otras bibliotecas (paquetes) auxiliares usando el comando “yarn”, aparte de las que ya vienen por defecto con el proyecto. Tener Yarn no es estrictamente necesario puesto que las bibliotecas también pueden instalarse con el comando “npm”, el cual queda automáticamente habilitado una vez instalado Node.js. Pero la ventaja de hacerlo con “yarn” es que el proceso suele ser mucho más rápido.

Para obtener Yarn nos dirigimos a su sitio oficial para descargas [2] y allí podemos escoger mediante una lista desplegable el sistema de operación para el cual queremos ver la explicación de cómo hacer la instalación. Al momento de escribir este documento, en la página sucedía que si estábamos en el idioma español, varios elementos de la lista salían en blanco, así que es preferible cambiarse al idioma inglés en caso de que queramos cambiar el sistema de operación, porque allí sí podemos ver todas las opciones (Figura 3).



**Figura 3**. Selección del sistema de operación en la página oficial de Yarn.

Cuando seleccionamos Windows, nos mencionan tres formas de proceder para la instalación. La más fácil es pulsar el botón “Descargar el instalador” (Figura 4). Una vez descargado el archivo, lo ejecutamos y seguimos las instrucciones.

****

**Figura 4**. Uno de los métodos para descargar Yarn para Windows.

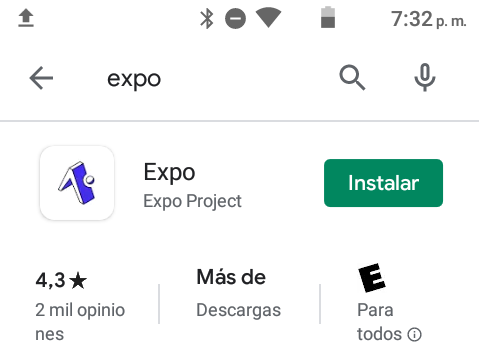
* **Interfaz de líneas de comandos (CLI, *Command Line Interface*).** Una vez instalados Node.js (y posiblemente Yarn), necesitamos habilitar la interfaz de líneas de comandos de Expo. Esto es lo que nos permitirá ejecutar una instrucción con el comando “expo”, como por ejemplo al hacer “expo start”. Para ello, en la terminal, estando en cualquier directorio, si queremos usar NPM podemos ejecutar “npm install -g expo-cli”. La bandera “-g” indica que la instalación va a ser global, es decir, que la nueva biblioteca podrá ser accedida desde cualquier directorio. También se pudo haber usado la palabra completa “--global”, en cuyo caso es necesario que esté precedida por dos guiones en lugar de uno. Por otra parte, si se quiere usar Yarn, ejecutamos “yarn global add expo-cli”, que como podrá notarse allí la palabra “global” no va precedida de guiones.

También se puede habilitar la interfaz de líneas de comandos de React Native con “npm install -g react-native-cli” o bien con “yarn global add react-native-cli”, que es la que permite ejecutar instrucciones con el comando “react-native”. Pero esa interfaz en realidad se utiliza cuando un proyecto se trabaja sin Expo, por lo que durante esta fase no se le dio uso.

Luego es posible que las rutas de las aplicaciones que se acaban de instalar todavía no se hayan añadido al “path” del sistema de operación, por lo que éste todavía no reconocerá los comandos “expo” ni “react-native”. En ese caso habría que añadirlas manualmente a dicho “path”, aunque también está la opción de reiniciar la computadora, y así ese proceso se hará automáticamente.

* **Android Studio**. Esto sólo es necesario si se desea emular la aplicación en la misma computadora con una máquina virtual en lugar de en un dispositivo móvil físico. El sitio oficial de Android Studio se encuentra en [3]. La emulación en computador no fue realizada durante la fase 3 de este proyecto, por lo que no haremos una descripción paso a paso de cómo se realizaría.

1. **Lo que necesitamos tener en nuestro dispositivo móvil físico de pruebas:** Allí sólo tenemos que instalarExpo. En el caso de que se esté usando un sistema de operación Android, abrimos la aplicación “Google Play Store”, que a veces aparece simplemente con el nombre de “Play Store” y normalmente viene instalada por defecto. Una vez dentro, en el buscador escribimos “expo”. Debe aparecer un resultado como el de la Figura 5, y le damos al botón de instalar:

****

**Figura 5**. Resultado de la búsqueda de “expo” en Google Play Store.

Si esta aplicación no aparece en los resultados, es porque el dispositivo no es compatible con ella y no será posible instalarla en él. Play Store se encarga de ocultar de los resultados las aplicaciones que no podrán ser instaladas en el dispositivo.

Por ahora sólo se ha comprobado la compatibilidad con versiones de Android 5 o superiores. En el sistema de operación iOS, el equivalente a Play Store es App Store.

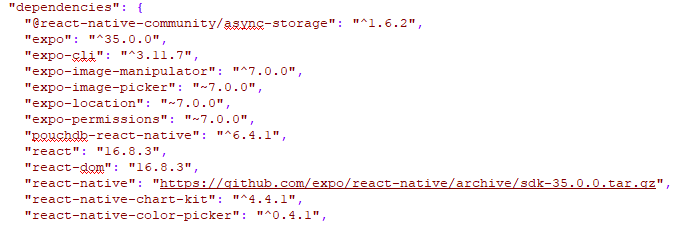
**Obtención del proyecto de la fase 3**

El proyecto tal cual se dejó en la fase 3 puede ser descargado como un comprimido .rar o .tar a partir de un enlace de Google Drive [4]. Esto incluye los códigos, los archivos “package.json” y “app.json”. El proyecto también se puede clonar de un repositorio de GitHub [5], aunque no hay allí más de un *commit* registrado, y por lo tanto no se puede ver un historial de versiones.

En esas direcciones se omitió la carpeta “node\_modules”, que es en la que están las bibliotecas utilizadas por la aplicación. La razón de no haberla incluido es que ocupa mucho espacio, y el contenido de ella puede volver a instalarse gracias a que la información necesaria está contenida en el archivo “package.json”. Por ello, una vez descargado el proyecto “lithodex”, lo primero que debe hacer el nuevo programador es ubicarse en el directorio raíz del mismo en la terminal y ejecutar el comando “yarn install” o “npm install”. Ello creará la carpeta “node\_modules”, y el mismo programa utilizará automáticamente el archivo “package.json” para saber qué bibliotecas debe instalar allí.

**Instalación de nuevas bibliotecas**

Ya se mencionó que los comandos “yarn install” y “npm install” instalan todas las bibliotecas que se encuentren escritas en el archivo “package.json”, el cual está en el directorio raíz del proyecto. Si se desea agregar una nueva, una forma de hacerlo es escribiendo directamente su nombre junto con su versión dentro de dicho archivo, y volviendo a hacer la instalación de todas las bibliotecas. Las bibliotecas añadidas normalmente se colocan dentro de la sección “dependencies”. Por ejemplo:



**Figura 6**. Parte del archivo “package.json”.

Nótese que el nombre se coloca entre comillas; seguidamente vienen dos puntos (:), y luego se coloca la versión a utilizar, también entre comillas. El símbolo (~) delante del número de versión significa que se busca una versión casi equivalente a la indicada, pero no necesariamente tiene que ser la misma. El símbolo (^) significa que tiene que ser compatible con la versión indicada, pero tampoco necesariamente la misma. También se pueden usar los operadores numéricos habituales de desigualdad, (<, >, <= y >=), para indicar una versión menor que, mayor que, menor o igual que, y mayor o igual que, respectivamente, en relación con la versión indicada. Si no se coloca ningún símbolo junto a la versión, como en el caso de “react” en la Figura 6, la versión que se instalará tendrá que ser exactamente la indicada.

Aparte de “dependencies”, el archivo “package.json” también tiene otras secciones, como “devDependencies”, en la que se colocan bibliotecas que sólo se desea que estén presentes durante el desarrollo de la aplicación pero no en su versión compilada, como las que permiten hacer pruebas unitarias.

Ahora bien, con los comandos “npm” o “yarn” también se puede hacer la instalación de la biblioteca específica deseada, tal como veremos a continuación:

1. **NPM:** La sintaxis es “npm install <nombre\_biblioteca>”. La palabra “install” puede ser reemplazada por “add” o simplemente por la letra ‘i’, ya que ambos funcionan como sus alias.
2. **Yarn:** La sintaxis es “yarn add <nombre\_biblioteca>”. Nótese que en el caso de agregar una biblioteca específica, la palabra “install” obligatoriamente tiene que ser sustituida por “add”, a diferencia de con NPM, en que se puede usar cualquiera de ellas.

Al terminar la instalación, el archivo “package.json” automáticamente quedará editado indicando la nueva biblioteca que se agregó a la carpeta “node\_modules”. Si lo que queremos instalar es una biblioteca global en vez de una local para el proyecto, agregamos la bandera “-g” o “--global” en el caso de NPM, o bien la instrucción “global” en el caso de Yarn, como ya hemos mencionado.

Tanto con NPM como con Yarn se puede especificar la versión colocando una arroba justo después del nombre de la biblioteca, sin dejar espacios, seguido del número de versión, como por ejemplo “yarn add react-native-view-shot@3.1.2”. Las reglas de los símbolos que explicamos anteriormente para indicar una versión cercana a la deseada, compatible con la deseada, mayor que la deseada, etc., también aplican aquí. Además, se pueden instalar varias bibliotecas al mismo tiempo dejando un espacio de por medio entre cada una, como por ejemplo “yarn add react-native-view-shot@3.1.2 react-native-fs@^2.16.2”.

Si el programador no indica el número de versión, estos comandos se encargan de elegir la más reciente. Sin embargo, a veces ocurre que esa más reciente resulta no ser compatible con otras dependencias que ya están instaladas, y eso aparece luego como un mensaje de advertencia cuando se va a emular la aplicación. Ese mismo mensaje indica cuál versión debería utilizarse.

Hay una diferencia entre NPM y Yarn que es importante señalar [6]. Con versiones de NPM anteriores a la 5, el comando “npm” tenía el inconveniente de que cada vez que se instalaba una nueva biblioteca, el orden en que se organizaban los paquetes y las dependencias que a su vez éstos tienen podía ser diferente cada vez, lo cual generaba algunos problemas, en especial si varias personas estaban trabajando en el mismo proyecto, porque cada uno podía tener la carpeta “node\_modules” en su respectiva computadora con una estructura distinta a las de los demás.

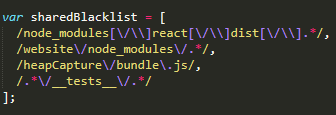
A partir de la versión 5, NPM implementó una solución a ello, añadiendo automáticamente un archivo de bloqueo llamado “package-lock.json” que indica cómo se hizo la instalación. De este modo, si el proyecto se va a compartir con varias personas, debe compartirse también ese archivo, y así cuando otro programador vaya a instalar los paquetes, NPM se guiará por el archivo y hará la instalación con la misma estructura indicada en él. En realidad, en versiones anteriores también era posible generar dicho archivo de bloqueo, pero debía hacerse manualmente a través de un comando, por lo que el programador tenía que recordarlo cada vez.

Por su parte, Yarn siempre implementó esa estrategia desde su primera versión, generando automáticamente un archivo de bloqueo llamado “yarn.lock”. La desventaja es que sólo garantiza que el orden sea el mismo en cada instalación si siempre se utiliza la misma versión de Yarn, a diferencia de NPM, que lo garantiza en cualquier versión superior a la 5. A pesar de ello, en esta fase se prefirió usar Yarn porque hace instalaciones mucho más rápidas, debido a su procesado por fases y en paralelo.

Durante esta fase se creó un código en Python llamado “postInstallation.py”, que está ubicado en el directorio raíz de “lithodex” y cuya finalidad es que el programador lo ejecute justo después de que haga una instalación de una o más bibliotecas. Para ello hay que dirigirse al directorio raíz de “lithodex” en la terminal y ejecutar la instrucción “python postInstallation.py”. (Nótese que esto requiere tener instalado Python en la computadora y que su ejecutable haya sido añadido al “path” del sistema de operación). Las razones de haber creado ese código son dos:

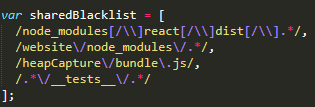
1. Durante la mayor parte del desarrollo de este proyecto ocurría una anomalía tanto cuando se instalaba una biblioteca específica como cuando se volvían a instalar todas las bibliotecas ejecutando “yarn install” o “npm install”. Esta anomalía impedía que la aplicación se compilara o se emulara, y según comentan en StackOverflow [7] ocurría por el hecho de que las versiones de Node.js posteriores a la 12.10.0 generaban un error después de una instalación. Es decir, esto era un problema de Node.js, no de este proyecto como tal.

Siguiendo las instrucciones de las respuestas que dan en [7], después de hacer la instalación había que cambiar manualmente una línea en el archivo “./node\_modules/metro-config/src/defaults/blacklist.js”. Dentro de dicho archivo, casi comenzando el código se encuentra una variable llamada “sharedBlackList” cuyo contenido debía ser como el que se ve en la Figura 7.

****

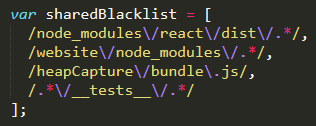
**Figura 7**. Variable “sharedBlackList” funcional.

El problema era que al hacer cualquier instalación, la variable cambiaba a como se ve en la Figura 8, en la cual se puede apreciar que en la segunda línea desaparecieron las tres barras ( \ ) de carácter escapado con las que inician los corchetes [ ]. Esto generaba problemas, y por ello el archivo “postInstallation.py” se encarga de corregir esa línea automáticamente.



**Figura 8**. Variable “sharedBlackList” modificada, no funcional.

Sin embargo, como ya se mencionó antes, durante el desarrollo de este proyecto las herramientas se han actualizado más de una vez, y con las versiones actuales la variable “sharedBlackList” aparece como en la Figura 9 después de hacer una instalación.



**Figura 9**. Variable “sharedBlackList” después de una instalación con las herramientas actuales.

No se ha determinado cuál actualización de cuál herramienta fue la que permitió que esa variable apareciera de esta forma, pero lo importante es que ahora la aplicación ya puede emularse directamente sin cambiar la línea, aunque de todos modos si se ejecuta el archivo “postInstallation.py” tampoco ocurre ningún error. Es decir, que tanto si aplicamos la modificación de la línea como si no, la aplicación funcionará.

1. El código “postInstallation.py” también corrige otro aspecto que evita que aparezca un mensaje de advertencia cuando se está emulando la aplicación. Ocurre desde que se actualizó la versión de Expo SDK a la 38. La razón es que las funciones que antes se invocaban como “ref.getNode()” ahora pueden invocarse directamente como “getNode()”, por lo que si ello no se cambia, Expo advierte que esto puede hacerse. El archivo que continúa invocando esa función de la manera antigua es: "./node\_modules/react-native-safe-area-view/index.js".

En conclusión, la funcionalidad del código “postInstallation.py” ahora es opcional, pero hubo un período en que era indispensable. Este código puede crecer en el futuro porque pueden seguir surgiendo otros errores o advertencias a medida que se actualizan las herramientas o se instalan nuevas bibliotecas.

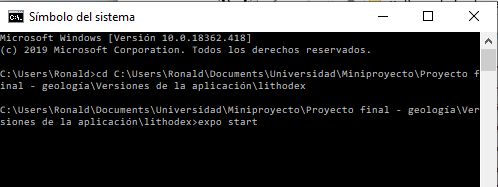
**Emular la aplicación**

Emular la aplicación significa que no se generará un archivo compilado sino que todos los códigos fuente se mantendrán en el servidor, es decir, en la computadora, y desde allí se permite que la aplicación se ejecute. Por ejemplo, se podrá correr en un dispositivo móvil físico diferente a dicha computadora. Emular es lo ideal cuando estamos desarrollando y queremos verificar rápidamente cómo se manifiestan los cambios que hemos hecho. A continuación explicaremos cómo hacerlo:

1. **Usando un dispositivo móvil físico:** Por ahora sólo se garantiza que la aplicación funcione en el sistema de operación Android, necesariamente de versión 5 o superior. No se han hecho pruebas en iOS. Es importante que tanto el servidor como el dispositivo móvil estén conectados a una misma red local, como por ejemplo usando la misma red WiFi, porque Expo utiliza las direcciones IP locales para referirse a los dispositivos terminales y, como debe saberse, dichas IP’s sólo son reconocibles dentro de la misma red. En esta fase del proyecto no fue investigado cómo emular si el servidor y el móvil no están conectados a la misma señal WiFi.

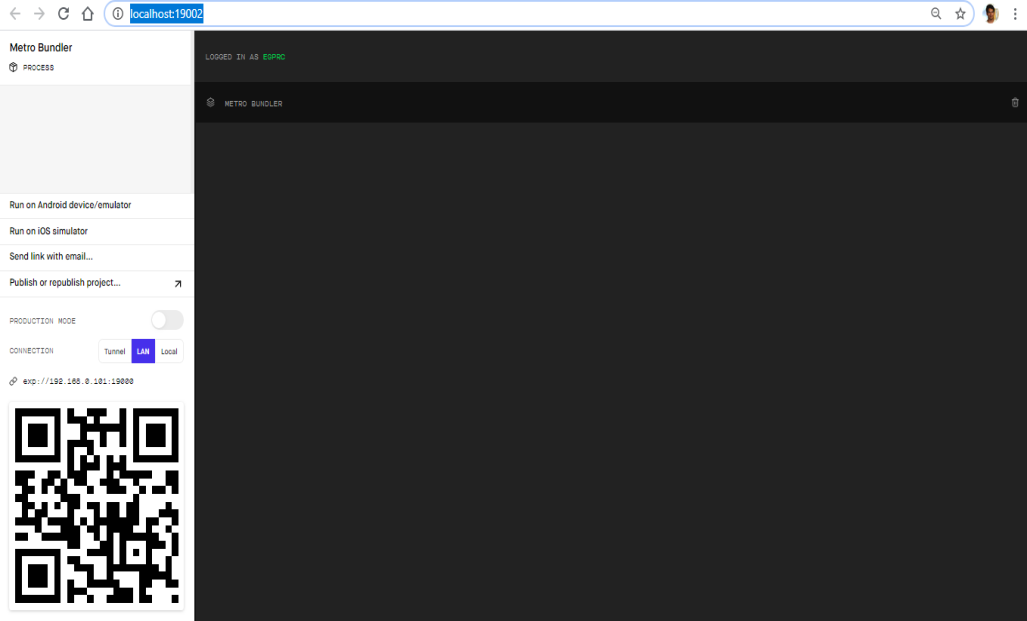
Como Expo suele usar los puertos 19000-19002 del *router* con el que se genera la señal WiFi, también puede ser necesario hacer una redirección de dichos puertos (*port forwarding*, en inglés) a la dirección IP local de la computadora. Hay múltiples maneras de hacer ello, dependiendo del modelo del *router* que se esté utilizando, por lo que no las mencionaremos aquí.

Una vez conectados, en la computadora del servidor nos dirigimos al directorio de “lithodex” a través de la terminal, y ejecutamos la instrucción “expo start”, tal como se ve en la Figura 10.



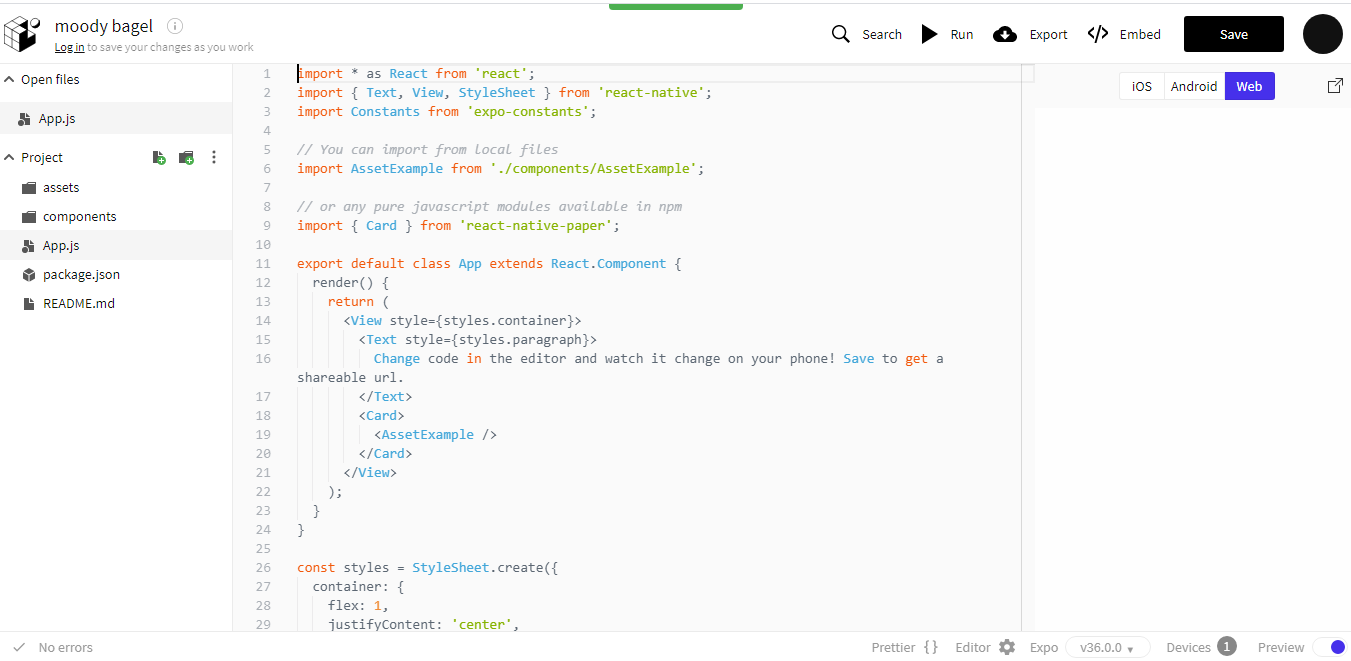
**Figura 10**. Ejecución de “expo start” en la terminal.

Eso abrirá posteriormente una página web como la de la Figura 11, en la cual aparece un código QR. En el dispositivo móvil abrimos la aplicación Expo, y presionamos sobre la opción “Scan QR Code”. Eso habilitará la cámara del teléfono y así podremos enfocar el código QR que se abrió en la página web de la computadora.



**Figura 11**. Ventana que se abre en un navegador cuando se ejecuta “expo start” desde la terminal.

1. **Usando un emulador en la web:** Sólo se ha encontrado el sitio web **Expo Snack** [8], que permite emular una aplicación que trabaje con Expo. Cuando lo abrimos, nos encontramos con un proyecto de ejemplo, el cual puede variar cada vez que accedemos de nuevo a la página. En la parte izquierda podemos ver los subdirectorios de ese proyecto, y en la parte central aparece el código del archivo que tengamos abierto (Figura 12).

****

**Figura 12**. Vista inicial de Expo Snack.

Eso nos debería permitir cargar allí nuestros archivos y luego emular la aplicación dándole al botón “Run”, que se encuentra en la barra superior con el símbolo universal de “Play”. El problema es que Expo Snack sólo trabaja con las bibliotecas que los desarrolladores de ese sitio web hayan añadido allí, por lo que nosotros no podemos instalar otras bibliotecas adicionales. Y como LithoDex usa varias bibliotecas que fueron agregadas posteriormente, como es el caso de 'react-native-elements' que no estaba en Expo Snack al momento de escribir este documento, entonces no pudo usarse la herramienta Expo Snack para emular LithoDex. El propósito de haberla mencionado es por si en el futuro llega a funcionar, y también para que el programador sepa a qué debe atenerse si intenta usarla.

En los foros de Expo escribieron en el 2017 [9] que todavía están trabajando para que Snacks pueda acceder a cualquier paquete “npm”, pero por los momentos sólo están disponibles los que mencionan allí.

1. **Usando un emulador de un sistema de operación móvil:** Esto se refiere a tener una máquina virtual del sistema de operación móvil en la computadora, y luego ejecutar allí la aplicación. En el caso del sistema Android hay que trabajar con la herramienta Android Studio, y en el caso del sistema iOS hay que trabajar con la herramienta Xcode. Sin embargo, esta modalidad no fue empleada durante la fase 3 de este proyecto. Se intentó ejecutar una versión compilada de LithoDex en los emuladores de Android en Windows: BlueStacks y MEmu, pero no hubo éxito.

**Mensajes de advertencia y errores al emular**

Ya se comentaron los problemas que surgen cada vez que se hace una instalación y que se pueden corregir con el archivo “postInstallation.py”. Hay otros que pueden ser solucionados permanentemente. Por ejemplo, fue necesario degradar la versión de “react-native-gesture-handler” a “~1.3.0” para evitar que cada vez que se emulara la aplicación apareciera la advertencia: “**Accessing view manager configs directly off UIManager via UIManager['getConstants'] is no longer supported. Use UIManager.getViewManagerConfig('getConstants') instead.**”

Hay otras advertencias que no se han podido eliminar, aunque ello no ha afectado la funcionalidad de la aplicación. Por ejemplo, ya no se recomienda usar la función “componentWillMount”, y por ello se evitó completamente su uso en los códigos de LithoDex. El problema es que algunas de las dependencias de los paquetes que están instalados en la carpeta “node\_modules” la siguen utilizando, y por eso aparece un mensaje de advertencia cuando se accede a ciertas vistas. Aprovechamos para mencionar que el propósito de esa función era indicar que se deben ejecutar ciertos procesos al momento de acceder a una vista, y actualmente para lograr el mismo objetivo hay que usar otras alternativas. En LithoDex se suele usar “componentDidMount”, que aunque no se activa exactamente en el mismo momento que lo hace “componentWillMount”, para efectos de esta aplicación no hay inconvenientes.

También aparece un mensaje de advertencia cada vez que se accede a alguna parte de la aplicación que utilice el componente “DatePicker” de la biblioteca ‘react-native-datepicker’. Ese mensaje indica que en un futuro el componente va a ser removido, por lo que recomiendan importarlo ahora del paquete ’@react-native-community/datetimepicker’, pero eso no se ha hecho porque el componente que viene en él tiene un formato distinto que no se ha sabido adaptar a las vistas de la aplicación.

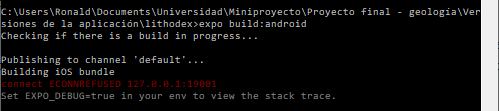
Otro mensaje dice: “**Cannot update during an existing state transition (such as within ‘render’). Render methods should be a pure function of props and state.**” Esto empezó a aparecer desde que se comenzó a utilizar una variable global, almacenada en la Tienda Redux (la cual se explica en el documento académico de este proyecto) para determinar si algunas vistas deben cargar datos o si ya no deben hacerlo. Lo que sucede es que algunas vistas necesitan leer primero cierta información desde la base de datos para poder mostrar su contenido de acuerdo a ella, como por ejemplo la que muestra la lista de afloramientos de un usuario particular. Mientras una vista está leyendo esa información, muestra un mensaje central que dice “Cargando”, y cuando ya ha terminado de leer, muestra el contenido propiamente. En algunas oportunidades fue conveniente indicar en una variable global de Redux si al regresar a una vista desde otra realmente se necesita volver a leer desde la base de datos o no, y por ello se incluyó esa variable global en la función “render”, que es la que se encarga de presentarle la vista al usuario. El mensaje de advertencia aparece porque si el programador no maneja esas variables correctamente, la aplicación podría entrar en un ciclo infinito, pero hasta ahora no ha habido problemas con eso.

Es frecuente que durante una emulación ocurra un error que la corrompa, que muestra un mensaje con fondo rojo que dice “Error: EPERM: operation not permitted, lstat…”. No se ha logrado descubrir la causa del mismo ni tampoco reproducirlo, pero puede suceder en cualquier momento, incluso si no se está haciendo nada. La solución ha sido detener la emulación y volver a iniciarla.

**Compilar la aplicación**

Aquí hablaremos de cómo compilar una aplicación que hace uso de la herramienta Expo, como es el caso de LithoDex. No mencionaremos cómo se compila un proyecto React Native puro porque no ha sido probado. Estando en la terminal, nos ubicamos en el directorio raíz del proyecto, “lithodex”. Si queremos compilar para Android, ejecutamos la instrucción “expo build:android”. (Nótese que entre la palabra “expo” y la palabra “build” hay un espacio, pero luego “build:android” se escribe sin dejar espacios). Por otra parte, si se quiere compilar para iOS, ejecutamos la instrucción “expo build:ios”, pero para ello es necesario solicitar a la compañía Apple una certificación que no es gratuita.

Normalmente, antes de ejecutar “expo build:android” en una terminal es necesario haber iniciado la emulación de la aplicación en otra terminal con el comando “expo start”, es decir, deben estar abiertas ambas terminales, ya que de lo contrario puede surgir el error “Connect ECONNREFUSED 127.0.0.1:19001”, que se señala en rojo tal como se ve en la Figura 13:

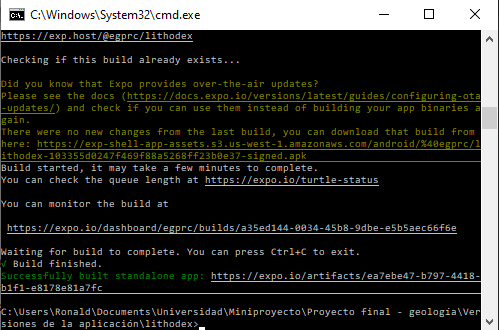


**Figura 13**. Error al intentar iniciar una compilación para Android con “expo”.

Si la compilación se completó satisfactoriamente, en el caso de Android se genera un archivo de extensión .apk, y en el caso de iOS se genera uno de extensión .ipa. La Figura 14 muestra la terminal después de una compilación finalizada para Android. El archivo se encuentra en la dirección que se indica donde dice en verde “Successfully built standalone app”.

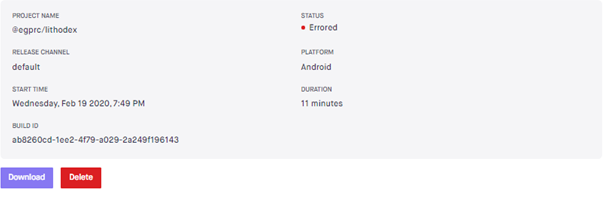
Si se copia y pega esa dirección en un buscador web, automáticamente se descarga el archivo. Para usarlo, es necesario descargarlo en el dispositivo móvil correspondiente y hacer la instalación. Es posible que el sistema pida una aprobación por parte del usuario, puesto que como este proyecto todavía no pertenece a una fuente verificada, lo verá como una posible amenaza.

Si ocurrió un error durante la compilación, entonces en la terminal se señalará en letras en rojo la sentencia “Standalone build failed!”. Son varias las razones por las que puede ocurrir un fallo, aunque si la aplicación lograba emularse, entonces lo más probable es que no se trate de un error de programación.

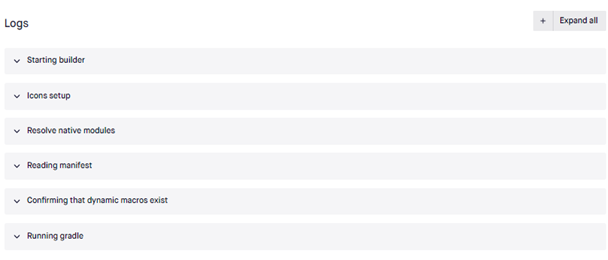


**Figura 14**. Compilación satisfactoria para Android.

En la Figura 14 también se muestra una parte que dice “You can monitor the build at”, seguida de un enlace. Ese mensaje puede aparecer incluso si una compilación termina fallando, pero sólo si primero hubo cierto progreso. No aparece cuando el error es temprano, como el ya mencionado de “Connect ECONNREFUSED 127.0.0.1:19001”. De modo que si queremos saber el motivo por el cual una compilación que parecía que iba a funcionar falló, nos dirigimos a ese enlace en un navegador web, y encontraremos una página como la representada por las Figuras 15 y 16.



**Figura 15**. Página de monitoreo de una compilación. En este caso se muestra una compilación fallida.



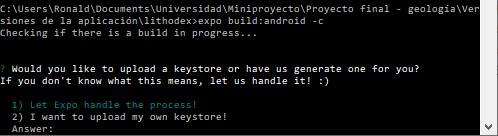
**Figura 16**. Página de monitoreo de una compilación (parte 2).

Nótese en la Figura 16 que hay varios documentos de registro (*Logs*, en inglés). Si se despliega el último (el que dice “Running gradle”), casi al final del mismo se señala el motivo del error, donde dice “What went wrong:”. En la Figura 17 dicho mensaje se ve la segunda línea, pero eso es sólo porque es una captura recortada de esa sección; lo que se ve allí es la parte final de toda la sección.



**Figura 17**. Sección final del archivo “Running gradle”.

En este ejemplo el error se debió a un problema con las credenciales de Expo, a saber, con el nombre de usuario y la contraseña. (Hubo que investigar el mensaje indicado en esa figura para averiguarlo). La solución fue añadir la bandera “-c” a la instrucción de compilación, “expo build:android -c”, ya que así Expo preguntará si se quiere cargar una clave ya creada (*keystore*), o si se desea que el mismo programa genere una nueva por nosotros. Se seleccionó dejar que Expo maneje el proceso (crear una nueva clave), la cual es la opción 1, que está señalada en azul en la Figura 18:



**Figura 18**. Caso en que se ejecuta el comando “expo build:android -c”

Sólo fue necesario realizar este proceso la primera vez que se realizó una compilación, puesto que las credenciales quedaron corregidas. A partir de allí, los siguientes intentos resultaron satisfactorios escribiendo simplemente “expo build:android”, sin “-c”.

Al principio de esta fase del proyecto fue posible compilar en Windows, pero después de actualizar algunas herramientas las compilaciones comenzaron a fallar. Desde entonces se trabajó con Ubuntu para hacer las compilaciones, a pesar de que el resto del desarrollo se continuó en Windows.

**Añadir permisos**

Por experiencia propia, es posible que algunas funcionalidades de la aplicación se ejecuten correctamente mientras se está emulando con Expo pero luego no lo hagan cuando se utilice la versión compilada. Las veces que ello ocurrió, se debió a un problema de permisos. Por ejemplo, ella necesita tener un permiso para acceder a la cámara del dispositivo y tomar fotos. Recuérdese que cuando se está haciendo una emulación con Expo, la aplicación que realmente se está ejecutando no es la nuestra sino Expo, y Expo sí tiene dicho permiso por defecto, a diferencia de la nuestra, a la que hay que agregárselo.

La Figura 19 representa un extracto del archivo “./app.json” en el que aparecen los permisos que hubo que agregar. Nótese que sólo los añadimos en la sección de “android”, porque es en donde hemos garantizado que funcionen; no se garantiza que todos tengan los mismos nombres para “ios”.



**Figura 19**. Parte del archivo “app.json”

Allí puede verse que en la sección “android > config > googleMaps” hay una clave llamada “apiKey”, que es necesaria para que la aplicación pueda integrar Google Maps, y que se obtuvo siguiendo los pasos indicados en la documentación de Expo [10]. Los otros permisos añadidos son:

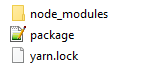
1. **ACCESS\_COARSE\_LOCATION:** Permite obtener la ubicación actual a través del proveedor de la red, que se basa tanto en las antenas de telefonía móviles más cercanas como en los puntos de acceso WiFi.
2. **ACCESS\_FINE\_LOCATION:** Permite obtener la ubicación actual del proveedor GPS, que hace uso de satélites, y según la documentación, también del proveedor de la red, al igual que lo hace “ACCESS\_COARSE\_LOCATION”. De este modo, si se tiene el permiso, en teoría ya no hace falta añadir “ACCESS\_COARSE\_LOCATION”. Sin embargo, según las pruebas que se realizaron, cuando éste se suprimió y se dejó únicamente a “ACCESS\_FINE\_LOCATION” en lugar de a ambos, la aplicación no fue capaz de obtener las coordenadas del punto actual (o quizás sí las obtuvo pero no las mostró cuando debería haberlo hecho, que es en los formularios de los afloramientos y los núcleos). Por ello se dejaron los dos permisos.
3. **LOCATION:** Se añadió porque en la documentación suelen incluirlo cuando una aplicación utiliza la biblioteca “expo-location”, como es el caso de LithoDex, aunque no logró determinarse que tuviera alguna ventaja con respecto a los dos permisos anteriores. De hecho, se probó suprimiendo este permiso y la aplicación siguió obteniendo correctamente las coordenadas de la ubicación actual. Pero se decidió dejarlo por si acaso resulta tener alguna funcionalidad adicional que no se logró descubrir.
4. **CAMERA:** Permite acceder a la cámara del dispositivo y tomar fotos con ella.
5. **CAMERA\_ROLL:** En teoría permite tanto leer como escribir en la galería de imágenes que tiene el sistema de operación por defecto, pero según se hicieron pruebas, también permite leer y escribir en cualquier otro directorio que esté ubicado en la memoria interna del dispositivo, no sólo en la galería.
6. **READ\_EXTERNAL\_STORAGE** y **WRITE\_EXTERNAL\_STORAGE:** Son los que respectivamente permiten leer y escribir en un sistema de almacenamiento externo que se le haya incorporado al dispositivo, como una tarjeta de memoria extraíble. Por ahora, LithoDex no puede escribir en una memoria externa porque no se ha logrado que al momento de salvar un archivo el usuario escoja el directorio en que desee hacerlo. Por ende, la aplicación siempre utiliza una ruta fija para hacer los guardados, la cual se decidió que fuera una que perteneciera a la memoria interna. Lo que sí pueden variar son las subcarpetas que emplea a partir de esa ruta fija. Pero por lo menos el permiso ya está agregado para cuando se logre hacerlo. Además, la documentación dice que cuando se indica el permiso “WRITE\_EXTERNAL\_STORAGE”, el “READ\_EXTERNAL\_STORAGE” ya queda garantizado, por lo que podría suprimirse.
7. **SYSTEM\_ALERT\_WINDOW:** Sirve para que una aplicación pueda mostrarse sobre otras. Hasta el momento este permiso no ha sido necesario, pero se deja por si en el futuro se quiere que muestre notificaciones de nuevos mensajes mientras el usuario está trabajando en otra aplicación.

Las explicaciones de éstos y otros permisos se pueden encontrar en el sitio para desarrolladores de Android “developers” [11].

**Creación de un nuevo proyecto React Native**

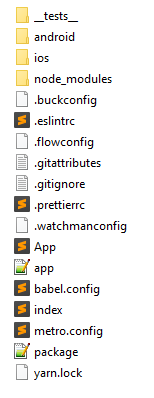
Idealmente un nuevo programador no debería tener que crear un proyecto nuevo React Native, sino que podría simplemente seguir trabajando sobre el que fue dejado en la fase 3. Sin embargo, es importante que sepa que hay varias maneras de comenzar un proyecto React Native desde cero, y que cada una creará una estructura distinta de carpetas dentro del mismo.

1. Una manera es ejecutando la instrucción “yarn global add create-react-native-app” o bien “npm install –g create-react-native-app” en la terminal. Eso descarga e instala de manera global la biblioteca que permite crear nuevos proyectos. Luego, en la misma terminal nos cambiamos al directorio dentro del cual deseamos colocar la carpeta que contendrá nuestro proyecto y ejecutamos “create-react-native-app <nombre\_proyecto>”. El “<nombre\_proyecto>” será el nombre de la carpeta contenedora. Dicha carpeta sólo tendrá lo que se muestra a continuación:



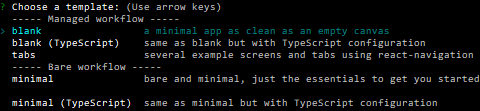
**Figura 20**. Resultado de la creación de un nuevo proyecto con “create-react-native-app”

1. Otra instrucción para iniciar un proyecto React Native es “react-native init <nombre\_proyecto>”. Nótese que en ese caso es necesario haber instalado la línea de comandos “react-native”. El proyecto resultante tendrá mucho más contenido que en el caso anterior:



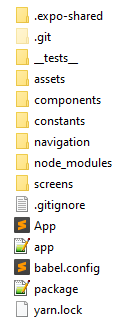
**Figura 21**. Resultado de la creación de un nuevo proyecto con “react-native init”

1. Una tercera manera es utilizando la instrucción “expo init <nombre\_proyecto>”. Si se omite el nombre del proyecto y solamente se coloca “expo init”, el mismo sistema solicitará el nombre posteriormente durante la instalación. Hay varios tipos de proyectos que se pueden crear, unos más complejos que otros. Dicho tipo también se solicita durante la instalación, como se ve en la Figura 22:



**Figura 22**. Opciones a elegir cuando se crea un proyecto con “expo init”

La Figura 23 muestra un proyecto creado con “expo init”, seleccionando la opción “tabs”, que trae varias carpetas más que las estrictamente necesarias, con la finalidad de proporcionar una aplicación simple de muestra. Por ejemplo, las carpetas “\_\_tests\_\_”, “assets”, “components”, “navigation” y “screens” tienen códigos de ejemplo que luego se pueden eliminar.



**Figura 23**. Resultado de la creación de un nuevo proyecto con “expo init”, opción “tabs”.

El proyecto LithoDex fue originalmente creado con el método 3, y por ello es que se puede emular y compilar con la aplicación Expo. Si se hubiese creado con alguno de los otros métodos, el proyecto sería React Native puro, y por ende la emulación habría tenido que realizarse de otra forma, como por ejemplo con una máquina virtual en Android Studio; o bien, si se deseara utilizar un dispositivo móvil físico, éste tendría que conectarse vía USB a la computadora, a diferencia de cuando se hace con Expo, que se escanea un código QR. Además, se tendría que utilizar la interfaz de líneas de comandos “react-native” [12].

Trabajar con un proyecto React Native puro tiene la ventaja de que permite incluir código nativo, en contraste a cuando se trabaja con Expo, que no se puede hacer, y en consecuencia la aplicación está limitada a las funcionalidades que vengan incluidas en el SDK de éste. Con esto no estamos diciendo que Expo no permita instalar nuevas bibliotecas, sino que sólo se puede con aquéllas que no utilicen código nativo. Además, al usar Expo no podemos escoger cuáles funcionalidades del SDK queremos mantener en el proyecto, sino que necesariamente tienen que estar incluidas todas, lo que significa que la aplicación ocupará un espacio adicional innecesario manteniendo funcionalidades que no necesitará.

**Expulsar Expo del proyecto**

Si por alguna razón se desearan añadir funcionalidades nativas a este proyecto, que fue comenzado con Expo, el programador puede ubicarse en la ruta del directorio raíz de “lithodex” en la terminal y ejecutar “expo eject”. Esto básicamente “expulsa” Expo del proyecto y lo transforma a modo parecido a como si fuese React Native puro, permitiendo añadir código nativo pero perdiendo muchas ventajas de Expo, como el poder emular la aplicación escaneando un código QR. Esta acción no es reversible. Sin embargo, el proyecto no se desliga totalmente de Expo porque se agrega un ExpoKit que permite seguir aprovechando las funcionalidades del SDK [13]. Por ejemplo, si el proyecto ya utilizaba una biblioteca propia de Expo, como puede ser “expo-file-system”, ésta debe poder seguir funcionando después de la expulsión. Pero debe tomarse en cuenta que durante esta fase no se comprobó que ello efectivamente se cumpla.

**Enlaces y referencias**

1. Sitio oficial para descargas de Node.js:

https://nodejs.org/en/download/

1. Sitio oficial para descargas de Yarn:

https://legacy.yarnpkg.com/lang/es-es/docs/install/#windows-stable

1. Sitio oficial de Android Studio:

https://developer.android.com/studio

1. Comprimidos del proyecto en la fase 3 en Google Drive:

https://drive.google.com/drive/folders/1ydw4Vh-82s1ggwVBNgHErPS3j\_uqIswQ?usp=sharing

1. Repositorio del proyecto en la fase 3:

https://github.com/RonaldBecerra/lithodex

1. Lijo, O. (9 de octubre de 2017). *Yarn vs NPM*. oscarlijo. Recuperado de:

https://www.oscarlijo.com/blog/yarn-vs-npm/

Consultado el 1 de mayo del 2020.

1. *How to resolve the error on ‘react-native start’* [Comentario en foro en línea]. StackOverflow.

Recuperado de:

https://stackoverflow.com/questions/58120990/how-to-resolve-the-error-on-react-native-start

Consultado el 03 de febrero del 2020.

1. Expo Snack:

https://snack.expo.io/

1. (Julio de 2017) *Modules available in Snack*. Forums. Recuperado de:

https://forums.expo.io/t/modules-available-in-snack/1651

Consultado el 16 de enero del 2020.

1. *MapView*. Expo. Recuperado de:

https://docs.expo.io/versions/latest/sdk/map-view/

Consultado el 17 de septiembre del 2020.

1. *Manifest.permission*. developers. Recuperado de:

https://developer.android.com/reference/android/Manifest.permission

Consultado el 25 de octubre del 2020.

1. *What is the difference between Expo CLI and React Native CLI?* [Comentario en foro en línea]. StackOverflow. Recuperado de:

https://stackoverflow.com/questions/54862388/what-is-the-difference-between-expo-cli-and-react-native-cli

Consultado el 26 de octubre del 2020.

1. *Ejecting to ExpoKit*. Expo. Recuperado de:

https://docs.expo.io/expokit/eject/

Consultado el 26 de octubre del 2020.