Installationsanleitung

Vorbereitung:

1. Installiere NodeJS https://nodejs.org/en/
2. `npm install exp --global` um Expo Funktionen von der Kommandozeile verwenden zu können.
3. Installation der Expo XDE: https://github.com/expo/xde/releases

   Bei Verwendung der XDE wird ein Login benötigt. Hierfür kann man einfach einen GitHub Account verwenden.

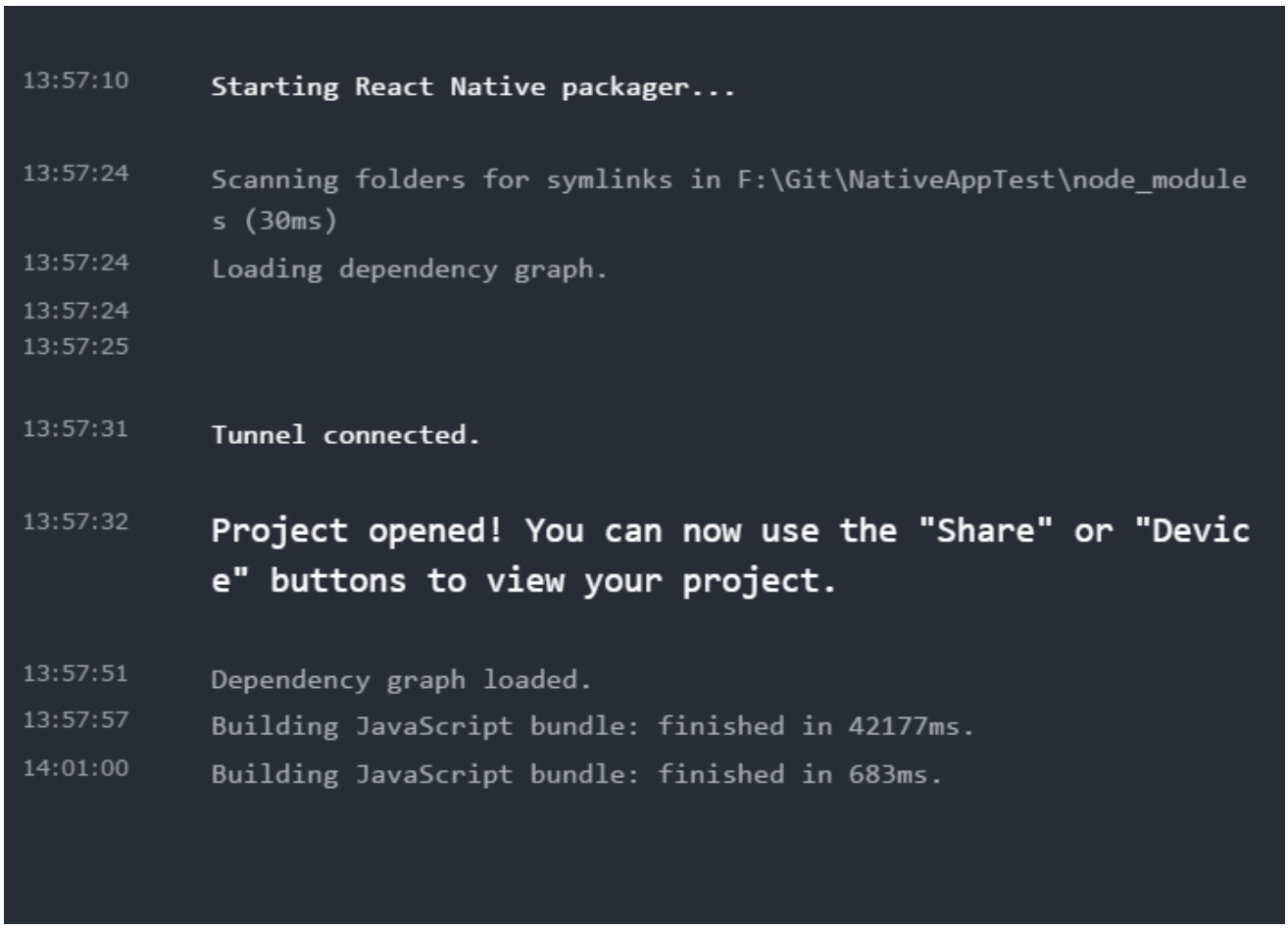
   Doku von Expo: https://docs.expo.io/versions/latest/guides/index.html
4. `npm install -g react-native-cli`
5. Wenn das Projekt von GitHub gepullt wurde muss im Projektverzeichnis der Befehl:

   `npm install` ausgeführt werden um alle in der package.json vermerkten Pakete zu installieren. Dies kann einige Minuten dauern. Bei mir kam es beim Installieren zu einem Problem wegen der dependency:

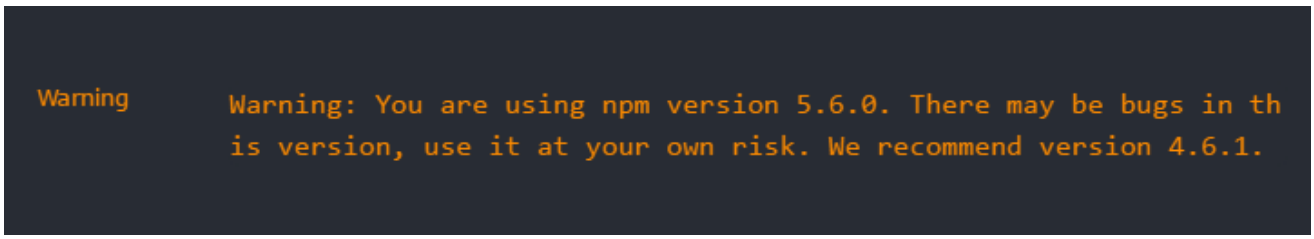
   `"react-native": "https://github.com/expo/react-native/archive/sdk-24.0.0.tar.gz"`

   Sollte hier ein Fehler auftreten einfach die Zeile mit der *React Native* dependency aus der package.json löschen und und den Befehl `npm install` ohne die react-native dependency nochmals ausführen. Nun sollte die Installation fehlerfrei durchlaufen. Nachdem alle Pakete installiert wurden, muss die Zeile wieder hinzugefügrt werden und `npm install` ein letztes mal ausgeführt werden.

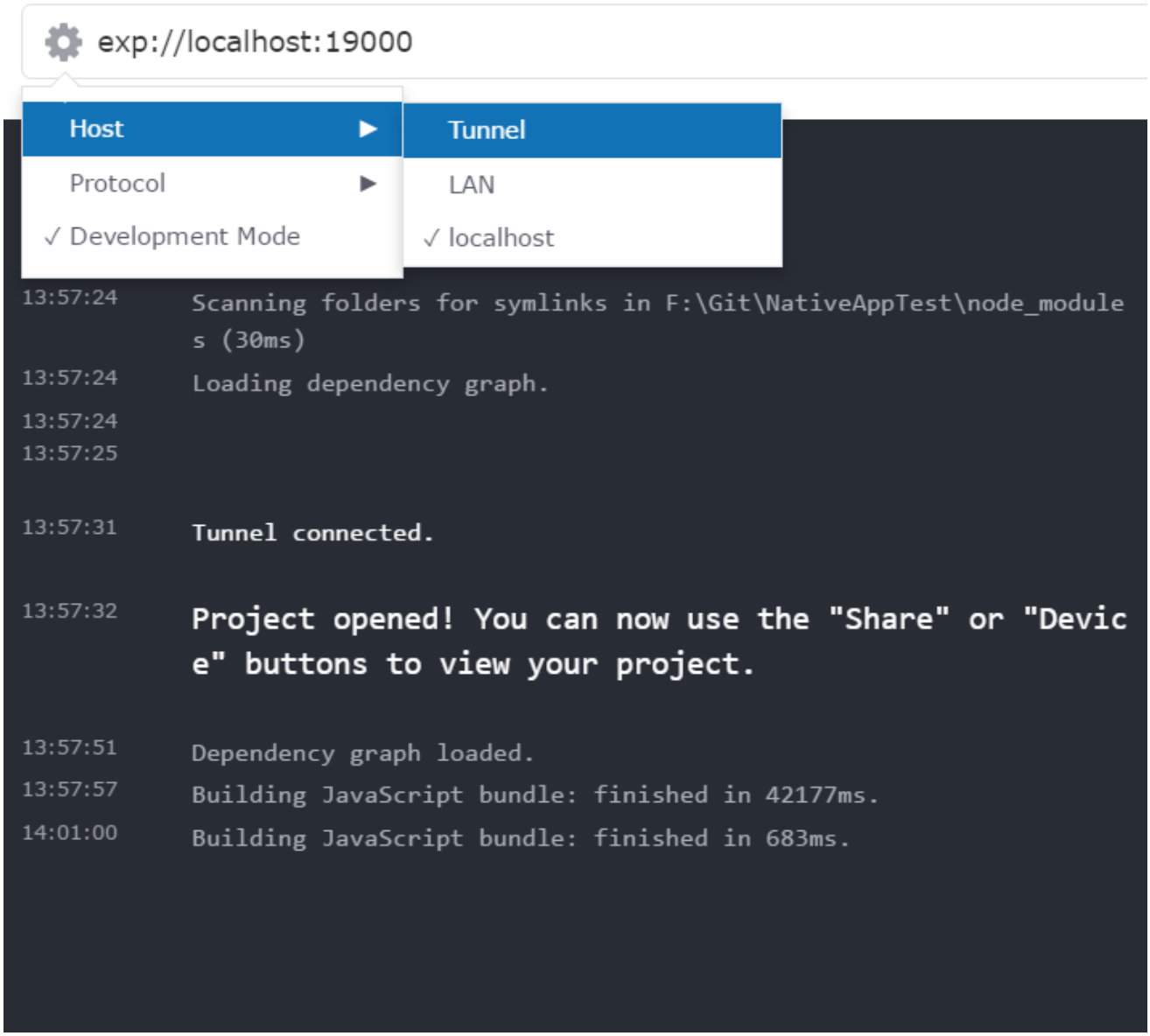
   Jetzt kann Expo XDE gestartet und der Projektordner ausgewählt werden.



Wenn *NodeJS* in Version v8.9.4 oder höher installiert wurde kommt npm in Version 5.6.0 mit, was zur Folge hat, dass EXPO XDE darauf hinweist, dass evtl. Bugs in dieser Version auftreten können. Bei mir hat dieser Hinweis zu keinerlei Einschränkungen geführt, sollte es jedoch zu Fehlern kommen empfiehlt es sich *NodeJS* in Version 7.10.1 zu installieren. Hier ist npm in Version 4.2.0 inkludiert.
https://nodejs.org/en/download/releases/



Mit einem Klick auf das Zahnrad kann der Verbindungstyp bzw. Host geändert werden. Wenn "Tunnel" ausgewählt wird, kann das Live Preview direkt auf dem Smartphone angesteuert werden.



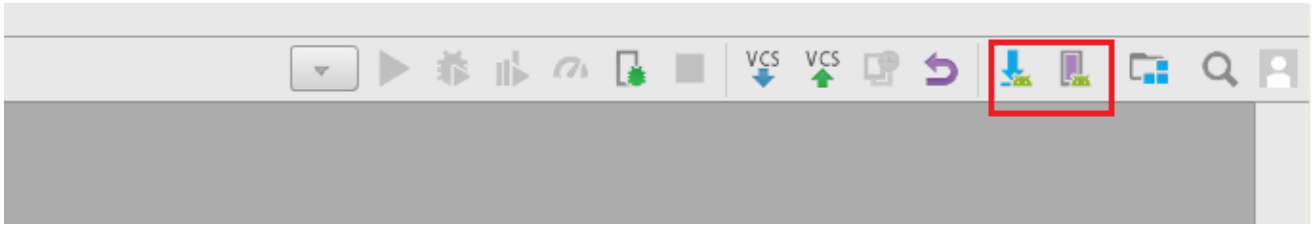
Einfach den Expo Client auf dem Smartphone installieren und den QR-Code scannen der angezeigt wird wenn man auf den Share Button klickt. Andernfalls die URL unter Explore in der Suchzeile der Expo App eingeben und bestätigen.

Wenn Android Geräte auf dem PC direkt getestet werden sollen, sollte *Android Studio* verwendet werden, da hierüber die Android SDK installiert wird und virtuelle Android Geräte erzeugt und verwaltet werden können. Die Installation kann einige Minuten in Anspruch nehmen. https://developer.android.com/studio/index.html

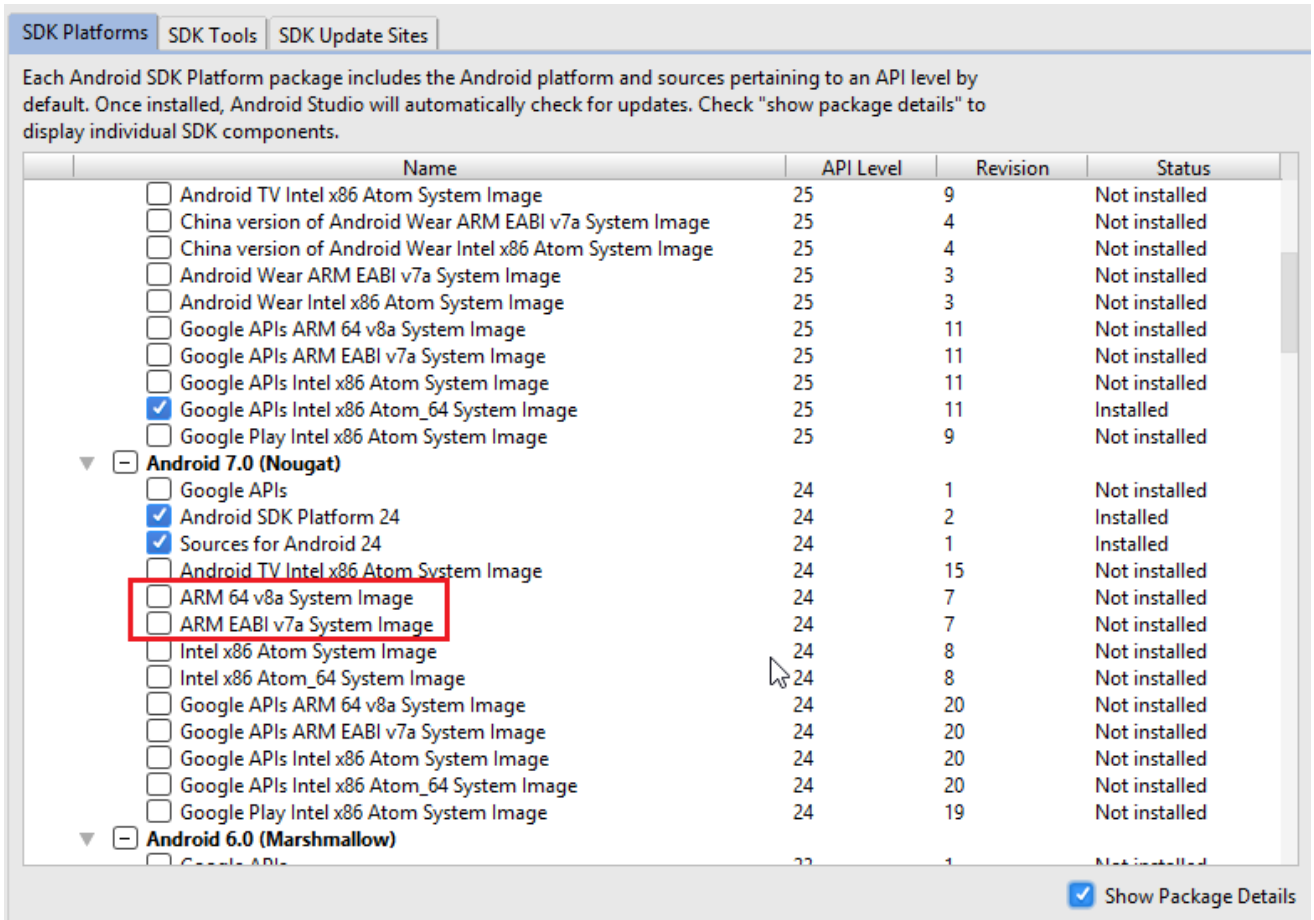
Da mit Expo bzw. *React Native* unabhängig vom OS gearbeitet wird empfiehlt es sich *Android Studio* erstmal nur als Quelle zum emulieren von Geräten zu verwenden. Ich erzeuge also nur ein separates Dummy Projekt um den AVD Manager nutzen zu können und importiere nicht das bestehende Projekt. Gradle erzeugt nun ein neues Android Projekt, das ungenutzt bleiben kann.

Wenn beim Start dennoch das bestehende Expo Projekt ausgewählt und geladen wurde wird beim ersten Laden des Projektes das Android Framewok erkannt und entsprechend konfiguriert. Im Projektordner sollten nun zwei neue Ordner mit Namen `.idea` und `gen` erscheinen die für Android genutzt werden könnten und für das *React Native* Projekt keine Bedeutung spielen. Da mit *React Native* kein Java erzeugt wurde kann mit Android Studio als IDE nicht gearbeitet werden.

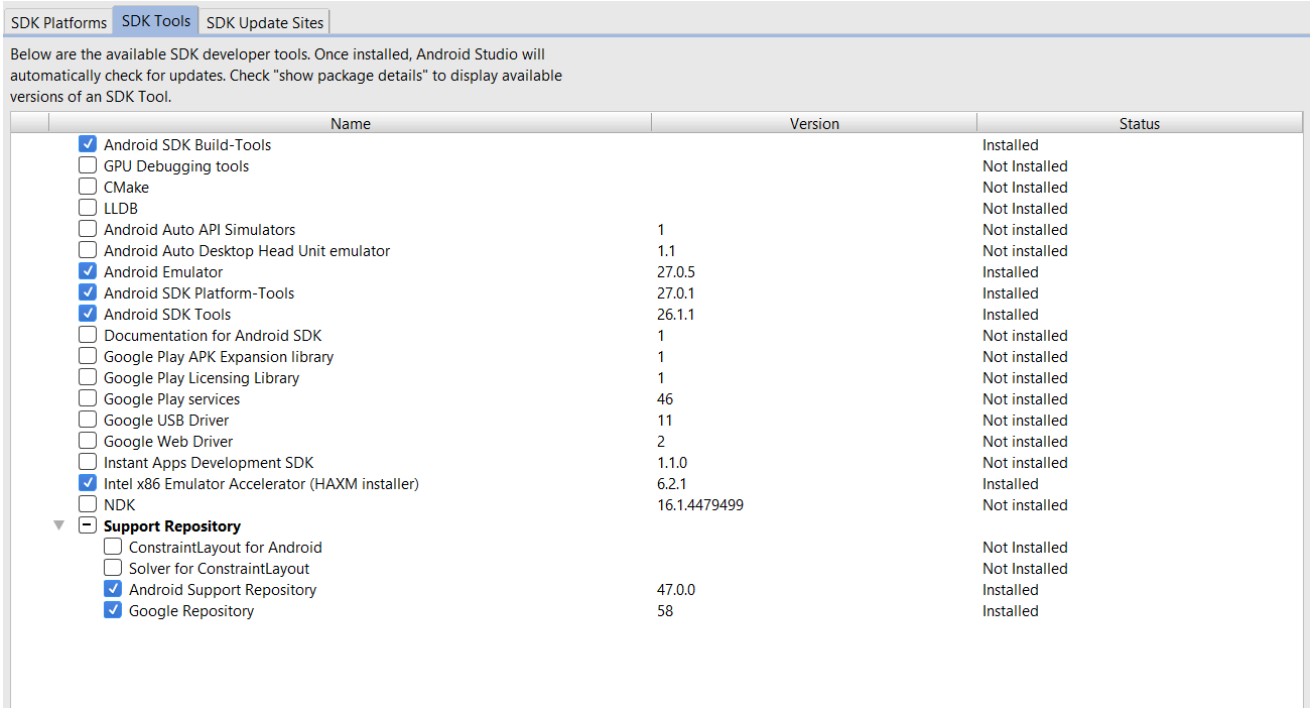
Im rechten Teil des Menübandes von Android Studio kann nun die SDK und der AVD Manager konfiguriert werden.



Über das Android SDK Menü können die enstprechenden SDK Platformen installiert werden. By default sollte die neueste Android Version ausgewählt und installiert sein. Klickt man auf das Häkchen bei "Show Package Details" kann man für die jeweilige Android Version auch ein ARM Package installieren. Diese Option ist wichtig falls man auf einem System mit einer AMD CPU (unter Windows) arbeitet und muss in diesem Fall ebenfalls installiert werden.



In dem Tab SDK Tools müssen die SDK-Build Tools, der Emulator, die SDK Tools und die Platform Tools zum installieren ausgewählt werden.



Der *Android Virtual Device Manager* erlaubt es nun virtuelle Geräte zu erzeugen. Es können vorgefertigte Hardwareprofile verwendet oder individuelle Profile erzeugt bzw. importiert werden. Nach der Auswahl des Geräteprofils muss im nächsten Schritt die entsprechende Betriebssystemversion auf dem Gerät installiert werden. Auch dies kann wieder einige Minuten dauern.

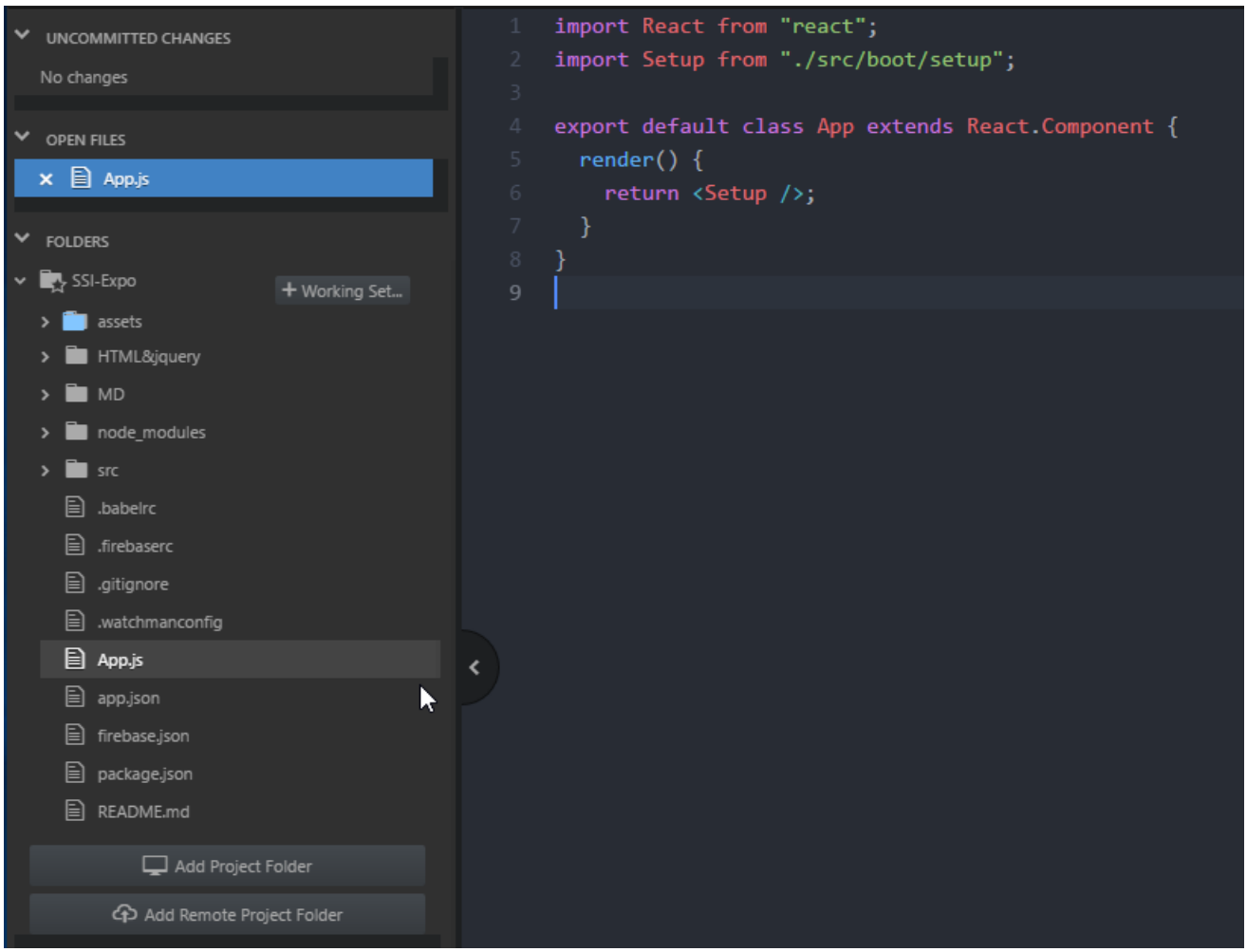
Wichtig ist hierbei, dass Computer (unter Windows) mit AMD Prozessoren nicht die Versionen in der Rubrik "Recommended" verwenden können. In dem Bereich "Other Images" muss eine entsprechende "arm" Version ausgewählt und installiert werden, damit das emulierte Gerät funktioniert. Das klappt nur, wenn vorher im SDK Manager das entsprechende ARM Package installiert wurde und auch dann läuft diese Variante nur sehr schlecht bzw. langsam und unzuverlässig. Unter Linux entsteht dieses Problem nicht.

Alternativ kann man Emulatoren wie *Genymotion* verwenden (kostenpflichtig).
https://www.genymotion.com/

Beim Emulieren von IOS Geräten auf dem MAC würde man die *Xcode Tools* verwenden. XCode stellt das entsprechende Pendant zu Android Studio dar und bringt eine IDE zur Entwicklung in Swift, Objective-C sowie C und C++ mit. Auch hier sollte man eine separate IDE verwenden da ReactNative von der *Xcode IDE* nicht unterstützt wird. Über den in den Xcode Tools mitgelieferten *iPhone Simulator* erfolgt dann das Erstellen von virtuellen Geräten. https://itunes.apple.com/de/app/xcode/id497799835?mt=12

Codestruktur:

Die Anwendung wird im Quellcode über die App.js gestartet, welche direkt im Quellverzeichnis liegt. Diese Struktur ist von *React Native* vorgeschrieben. Die Datei repräsentiert quasi den Einstiegspunkt der App. In diesem Fall verweist die App.js auf eine setup.js Datei in einem Unterverzeichnis.

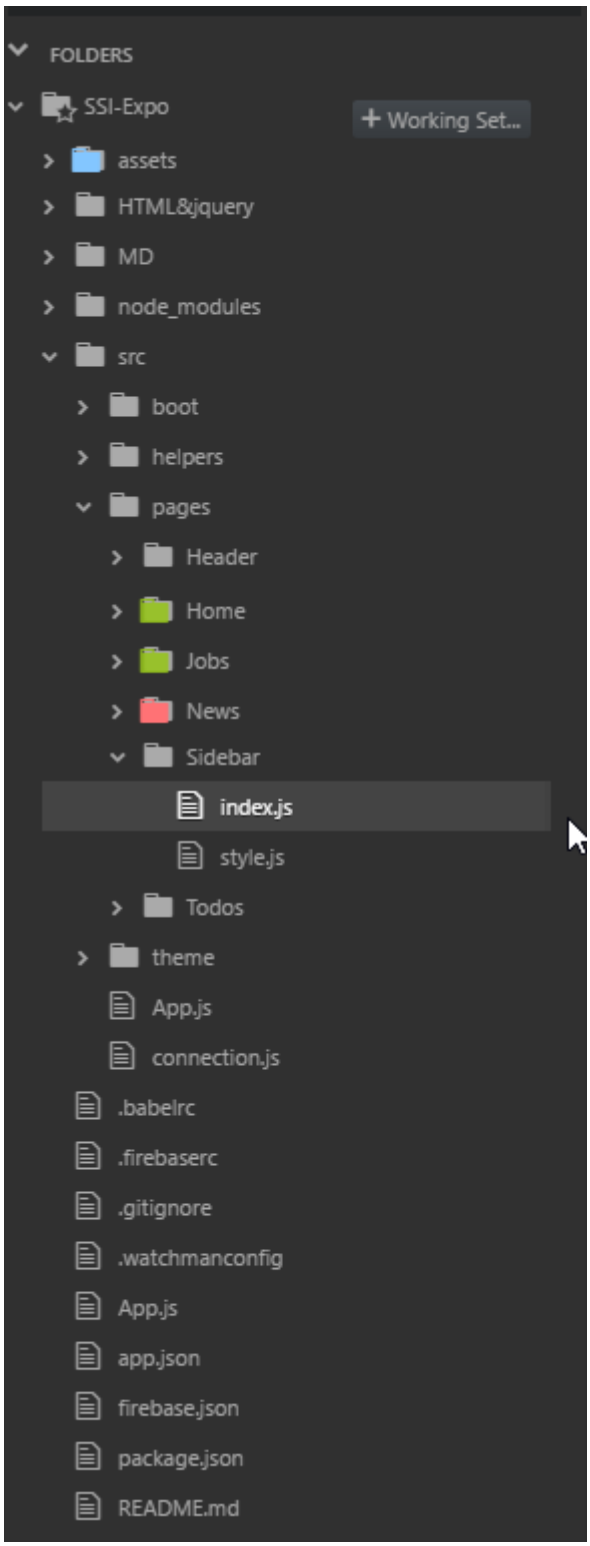


Die setup.js konfiguriert wichtigte Style Elemente die zur Ausführung der App notwending sind. Zum einen wird die Roboto Schriftart vorgeladen und zum anderen wird ein vordefiniertes Theme aktiviert. Das "Vorladen" der Schrift ist wichtig, da im Folgenden das Framework *NativeBase* verwendet wird, welches diese Schriftart benötigt. Wir die Schriftart nicht vorgeladen kann es leicht zu Fehlern kommen da Roboto nicht zu den Standard Schriftarten gehört.

Von der setup.js wird eine weitere App.js angesteuert, die sich direkt im Verzeichnis src befindet. Während der Dateiname der App.js im Hautptverzeichnis vorgegeben ist und sich nach den Konventionen von *React Native* richtet ist diese zweite Verwendung des Dateinamens "App.js" frei gewählt und kann jederzeit geändert werden.

In dieser zweite App.js wird die Navgiationsstruktur der App festgelegt. Es wird eine Initial Seite übergeben, ein StackNavigator initialisiert und vor allem ein Drawer definiert. Über diesen Drawer erfolgt später vornehmlich die Navigation durch die App. Das Aussehen und Verhalten des Drawers wird über die Sidebar Klasse reguliert. Jeder Screen der Anwendung wird in der App.js vermerkt damit die Navigation durch die App im Folgenden fehlerfrei funtioniert. Dem StackNavigator werden die jenigen Screens zugeordnet die später per Klick auf ein Element bzw. durch ein Event direkt angesteuert werden. Der Drawer wiederum enthält die Hauptscreens der Anwendung.

Alle Screens werden durch eigene Klassen repräsentiert und befinden sich in der Ordnerstruktur des Projektes in jeweiligen Unterordnern innerhalb des Ordners "/src/pages/". React Native steuert, wenn kein Dateiname sondern nur ein Pfad als Verweis angeben wurde, immer die index.js in einem Ordner an. Daher ist die Ordnerstruktur auch entsprechend so aufgebaut, dass sich in fast jedem Unterordner eine index.js Datei befindet. Die style.js enthält dann immer die jeweiligen, spezifischen visuellen Anpassungen.



Tokens für Push-Nachrichten:

Die App enthält einen Bereich Login, der aktuell dazu dient Geräte für Push Nachrichten zu registrieren. Sobald sich der Nutzer registriert wird in Firebase ein Account angelegt. Aktuell ist in Firebase das Verfahren Authentifizierung mit Email und Passwort aktiviert. Hierbei übernimmt Firebase die Authentifizierung komplett. Firebase verwendet den bcrypt Algroithmus um das Nutzerpasswort zu hashen. https://www.firebase.com/docs/web/guide/login/password.html Im Authentication Bereich von Firebase werden pro Nutzer so Email, eine unique ID, ein Erstellungs- und Anmeldedatum gespeichert. In der noSQl

Database von Firebase wird dann, wenn der Nutzer dem zustimmt, ein neuer Eintrag erstellt. Jeder UID kann so als neuer Eintrag die entsprechende Email Adresse des Nutzers und ein Token zugeordnert werden. In diesem Fall handelt es sich bei dem Token um einen ExpoToken da später über den Expo Sever Push Nachrichten versendet werden. Allerdings könnte man hier auch einen anderen Dienst verwenden bzw. die Android und IOS Dienste benutzen.

In dem Ordner "HTML&jquery" befindet sich ein Lösungsansatz (Verwendung von jquery und ajax) zum Versenden von Push-Nachrichten an alle in der DB hinterlegten Expo Tokens. Hierzu muss man sich zu erst per Auth bei Firebase verifizieren und erhält so Zugriff auf die Tokens. Das senden einer Push-Nachricht ist ein einfacher Post Befehl der an den Expo Sever https://exp.host/--/api/v2/push/send abgesetzt wird:

```
[
    {
        "to": "ExponentPushToken[jxHkxGHPpOqXFeZGVYlabc]",
        "title": "Test Nachricht",
        "body": "Hello World!"
    }
]
```

Bei Versenden von Push-Nachrichten wird also immer auf die gleiche Art und Weise vorgegangen. Der Token dient als eindeutige Identifikation des Gerätes, der Expo Server als Vermittler - für das Senden von Push Nachrichten benötigt man also nur den jeweiligen Token des Gerätes - hierzu sind keinerlei Berechtigungen von Nöten. Daher sollten die Tokens auch entsprechend gut gesichert werden.

Wenn die App bzw. beim Testen der Expo Client auf dem Smartphone deinstalliert wurden ist es nicht mehr möglich an den Token dieses Nutzers Nachrichten zu senden. Wird die Anwendung erneut installiert und der Nutzer loggt sich mit seinem alten Login wieder ein, wird ein neuer Token erzeugt.