BLD125417

Create iPhone Apps Using Forge Viewer & React Native

Michael Beale
Autodesk, Inc

Learning Objectives

- Learn about build tools for native app development using your existing web skills
- Learn how to develop 'Hello world' on your iPhone / Android device without XCode
- Learn how to add a navigation UI for browsing “files, folders & projects”
- Learn how to connect navigation UI with Forge Model Derivative API
- Learn about 3-legged Authentication on mobile
- Learn how to display a headless “2D/3D Forge” viewer inside your app
- Add a custom native UI Toolbar
- Learn about how to add offline support

Introduction

Improving communication between the job-site and the office is key to many things - improving project schedule, addressing safety issues and ultimately improving the bottom line.

Using the old paper and timesheet, while still significant, is being used in conjunction with iPhone’s and iPad app’s. These apps bring BIM/manufacturing data onto the job site, and job-site data back to the office, in real-time and offline.

You can also see this trend - just look at the number of mobile apps that target the BIM industry - each solving a particular business case.

Whether you need an app to snap photos of Issues/RFI’s, an app to do timesheet management, conduct a safety meeting, do sign-off inspection or punch-list creation - “there is an app for that”.

If you are planning on building a mobile solution for your business, then at some stage, you will need to figure out how to connect into your business data. The Autodesk Forge platform is the bridge to connect your App to your data.

The Forge Platform and Mobile

Autodesk Forge platform taps into the rich data sets of the Autodesk ecosystems - BIM-Team, Fusion-Team, Calloration for Revit, Autocad, Navisworks, Infraworks, Recap, Fusion/Inventor, Issues API and others. Mobile apps have the opportunity to tap into that rich data, but also contribute to these data streams. Mobile can make these rich data sets accessible on the field and job-site as well provide ‘chat’ tools specific to your industry or teams.

The target audience for mobile is typically for AEC or manufacturing businesses that have already integrated their databases with the Forge platform, but are now looking at mobile solutions to extend data-flow to the job-site, or new chat tools to their customers.

This instruction demo will show you how to get started with connecting these Forge data streams into a mobile app, using the React Native framework - so if you are familiar with web development, you already have the skills and web tools you need to get started.

The end result is a simple app that lets you login into the Autodesk network, such as A360docs, and browser through your existing projects and 2D & 3D files and then do some basic design-review operations, like 3D sectioning, with the Forge Viewer.

We will start from scratch, with a ‘hello world’ example to get your familiar with the new mobile development cycle.

Once familiar with the development cycle, we will extend ‘hello world’ with some UI, I call this the ‘shell’ app, since it has no data. Then I will show you how to wire up the shell app to the Forge APIs to display a login, some files and a 2D/3D view. This will touch on the Forge’s 3-legged authentication, the Model Derivative APIs and the Forge Viewer. This class is not suitable for developers at the beginner level.

Finally, I will wrap up with pointers on where to extend the app - for things like adding a ‘chat’ component, taking photos, notifications on the lock-screen and 3D-Markup with Apple’s new ARToolkit.

Your Forge DevCon Expert(s)

Michael Beale *is a Senior Developer Consultant since July 2017 for the Autodesk Developer Network and Forge Development Partner Program. Before joining the Forge Platform Team, Michael worked on Autodesk Homestyler, Cloud Rendering, Stereo Panorama Service and A360 Interactive Cloud Renderer before working on the ‘Forge Viewer’ and Viewing APIs. Originally from Australia, he believes that camera Z-up should be renamed ‘camera Z-down’.*

Developing mobile apps connected to Forge

I've been interested in React Native for a while, and have been wanting to use it to build a mobile version of the popular *three-legged NodeJs Forge sample code.* The biggest challenge here, is 'Authentication on mobile'. We want our phone to remember our login info, and not bother us again (until we uninstall or sign-out).
The end result of this, is a template called 'Forge Mobile App'. You can find the GitHub repo here (github.com/wallabyway/forgeApp).   It connects to the Forge API and gives you the basics for building native UI with React Native

What is React Native?

React Native, or RN for short, is developed by Facebook and has a vibrant React community. It continues to trend in popularity and many companies like Walmart, AirBnb, Instagram, Facebook and Oculus (React VR) are building flagship app's with React Native.

React Native is a flavor of React, so if you are familiar with React development, then you'll be right at home. In particular, we are leveraging the new 'Create React Native App' or CRNA and using Expo App to test things.

Why not use Swift/Java or Ionic ?

Forge is happy to integrate with any platform.

React Native Apps are written in javascript, where the JSC/V8 engine creates & manipulates Native UI components for both iPhone and Android platforms. This gives your app speed and the buttery smooth animation, of a typical native app. Native apps typically involve two separate development teams, one for iPhone (Xcode/Swift/Objc) and one for Android (A-Studio/Kotlin/Java), which also involves a higher development cycle cost, than React-Native's single code-base.

Another popular alternative is Cordova (and it's offspring Ionic) which uses a webBrowser to display the UI. However, Cordova/Ionic frameworks, while easy to use, typically suffer from 'animation jank', Navigation bugs, keyboard popup bugs, Date-Time picker issues, Form input and memory bloat. Their HTML UI tries hard to emulate the iPhone or Android UI, but it never quite looks or feels quite right.

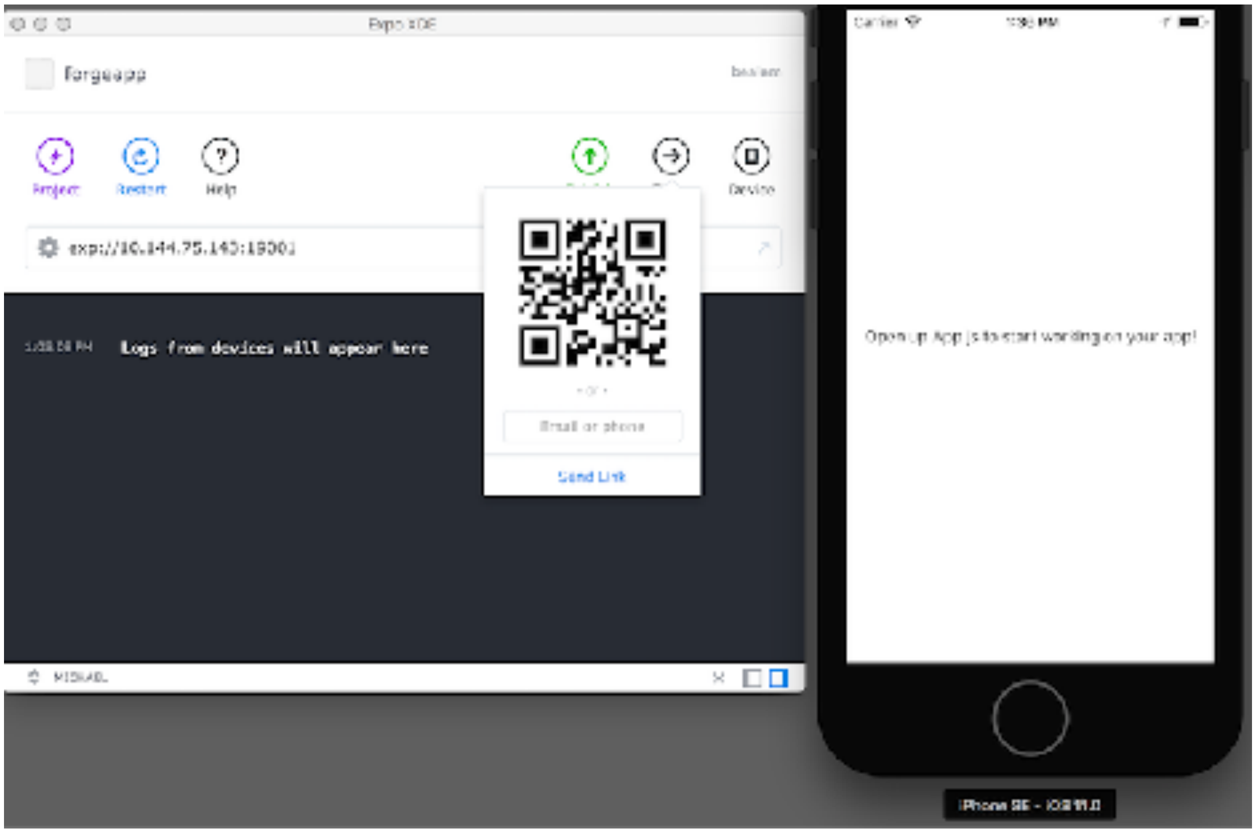
React Native is lighter weight (less memory, smaller file size) and has better overall performance than Cordova/Ionic. There is less effort in getting the UI correct, since RN uses native UI.

So let's take a look… Let's start with 'hello world'

1. Getting Started - Hello World

Setup

1 Install 'Expo' on your Android or iPhone (go to the App store)
2 install Expo's desktop tool 'XDE' on your Mac / Windows computer
3 Launch XDE
4 click 'Create New Project', use 'Blank' template
5 Type in project name 'forgeapp' and click 'Create' (see video here: https://youtu.be/2D8uzNYwdnY)



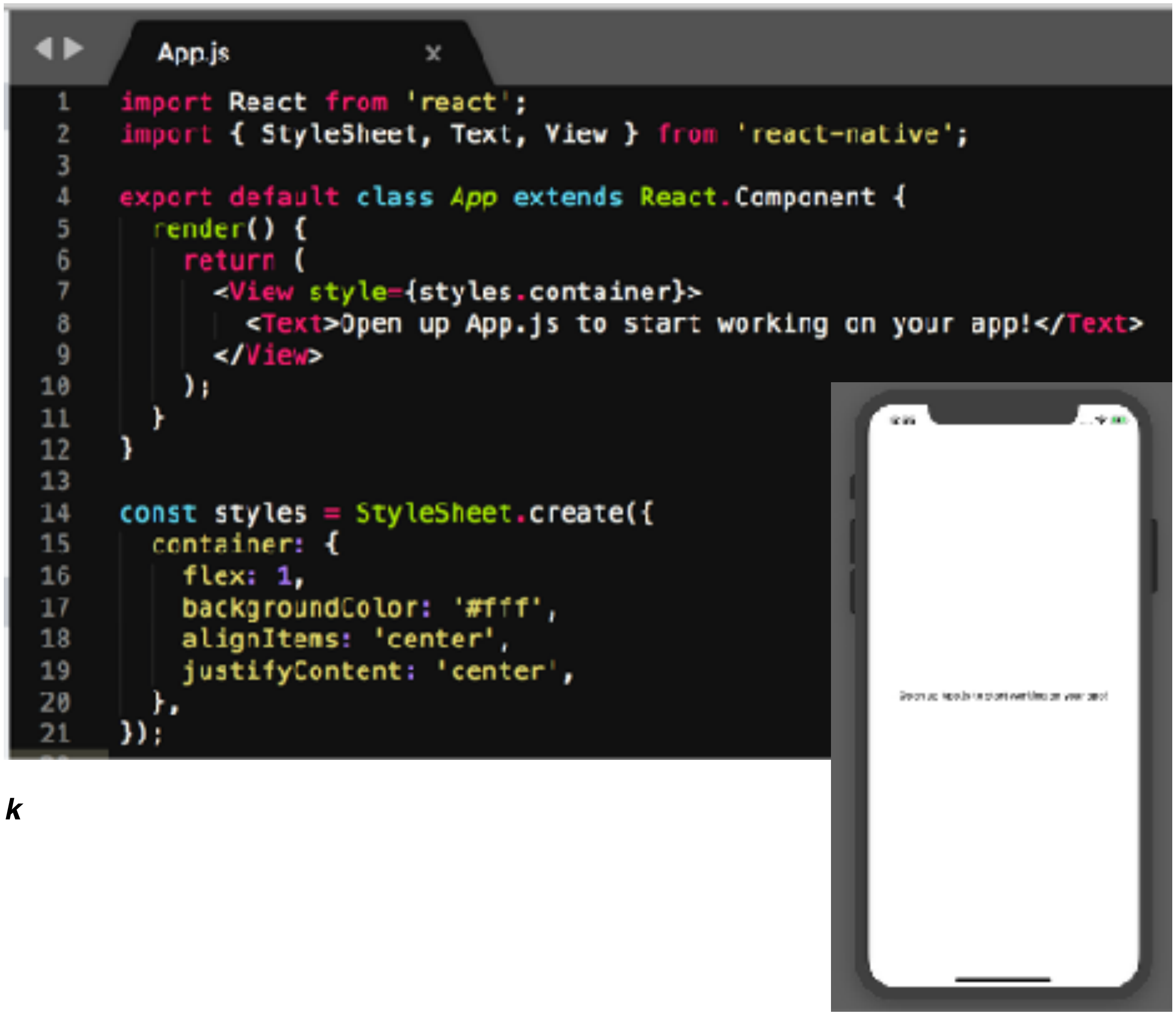
```
exp://10.144.75.140:19001
Logs from devices will appear here
```

Open up App.js to start working on your app!

6. click the 'settings' button, and select 'LAN' (see the video)
7. Click 'Share' to see a QR-code
8. Now scan the QR Code with the Expo app on your iPhone/Android (you can see me demonstrate this in the video from my slides)

Let’s look at 'App.js'

This is the entry file for our App. It is standard react code, with the difference that <DIV> elements are named <VIEW> and <TEXT> in react native. The React Native Introduction has more details.
Now, if you edit the text message in App.js, with a text editor (say VSCode or Sublime Text 3) and click save, your phone will automatically update with the change.
Note: Your phone and computer should be on same Wifi network. If you have Xcode installed, you can use the IOS simulator and work without a network (& offline), but note that 3D performance is slower.

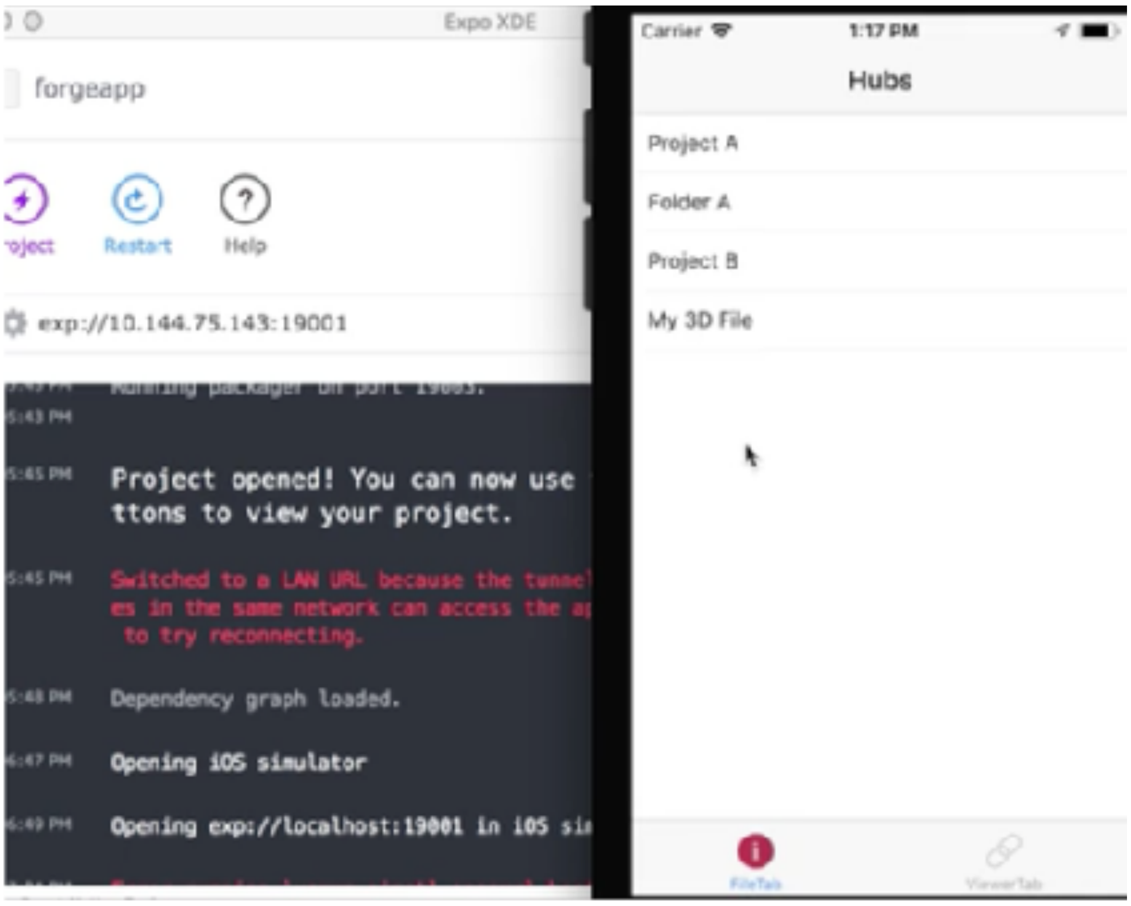


```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

export default class App extends React.Component {
  render() {
    return (
      <View style={styles.container}>
        <Text>Open up App.js to start working on your app!</Text>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

k

2. Adding UI - The 'Shell App'

Next, let's add some UI. Our app needs some kind of 'file browser' and a '2D/3D viewer', so we'll use the classic iPhone tabs metaphor to switch between these two 'Views'.   This will be our 'shell app' UI and we can use 'dummy data' to get us started.



```
5:43 PM
5:45 PM  Project opened! You can now use
         ttons to view your project.
5:45 PM  Switched to a LAN URL because the tunne
         es in the same network can access the a
          to try reconnecting.
5:48 PM  Dependency graph loaded.
6:47 PM  Opening iOS simulator
6:49 PM  Opening exp://localhost:19001 in iOS si
```

Here are the three UI components we'll use:

1. React-Navigation (Tabs and Views)
2. FlatList (List of files)
3. WebView (Forge Viewer)

The react-navigation component is what glues together, the 'Tab' bar at the bottom, the file browser (FlatList) and the Forge viewer (webView).

Setup

1. Copy these files from gist - App.js, js/fileList.js, js/styles.js, js/viewer.js (https://gist.github.com/wallabyway/1f53cc86cab2f5de1bc2eaa364503168)
2. install **react-navigation** component...ie.

in XDE, click project, open in terminal, and type...

```
> npm install react-navigation
```

3. Now refresh your phone (Click share, Scan QR code)

You should see the result in the video above.

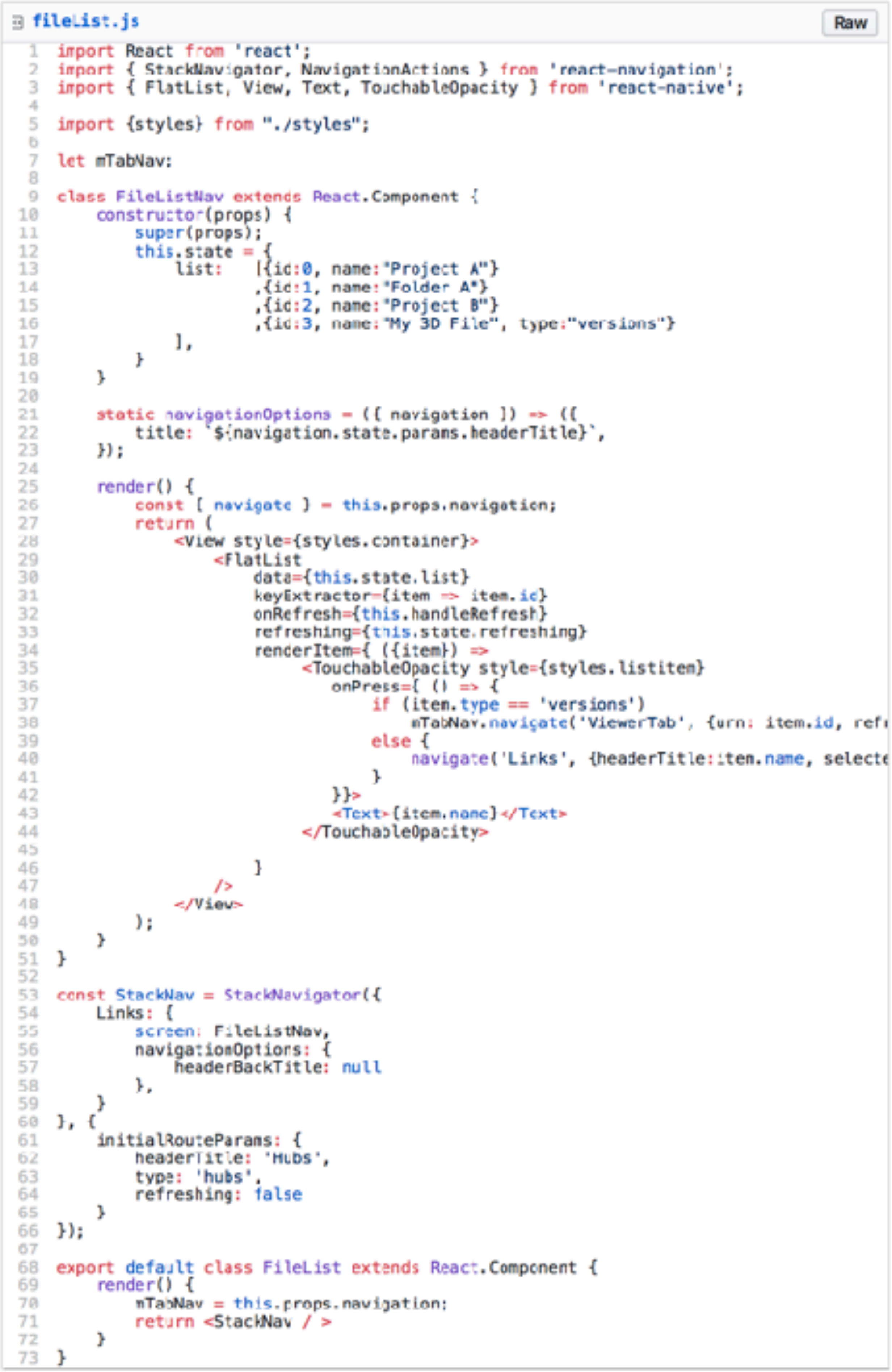
Here's what's going on:

We have the following new files:
App.js - this is our main entry file and contains a new 'tab navigator'
js/fileList.js - this is the stack-Navigator, and it holds a List View
js/viewer.js - this is the Forge Viewer displayed in a webView
js/styles.js - this is a single file to define our UI color-scheme etc

(refer to: https://gist.github.com/wallabyway/1f53cc86cab2f5de1bc2eaa364503168)

fileList.js - Stack Navigation

The fileList.js file does two things - it renders a listView and navigates Views. You can see in lines 13-16 the dummy JSON data that the FlatList UI component is rendering. When you click on a list item, an event triggers line#36 onPress method which navigates, or 'pushes the next page view' onto the stackNavigator. Clicking the back pops the View off the stackNavigator without code. We will use listView and StackNavigator to build our files/folders tree structure. You can read more about Stack Navigation here: https://reactnavigation.org/docs/navigators/stack



fileList.js

Raw

```
import React from 'react';
import { StackNavigator, NavigationActions } from 'react-navigation';
import { FlatList, View, Text, TouchableOpacity } from 'react-native';

import {styles} from "./styles";

let mTabNav;

class FileListNav extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            list:   [{id:0, name:"Project A"}
                    ,{id:1, name:"Folder A"}
                    ,{id:2, name:"Project B"}
                    ,{id:3, name:"My 3D File", type:"versions"}
            ],
        }
    }

    static navigationOptions = ({ navigation }) => ({
        title: `${navigation.state.params.headerTitle}`,
    });

    render() {
        const { navigate } = this.props.navigation;
        return (
            <View style={styles.container}>
                <FlatList
                    data={this.state.list}
                    keyExtractor={item => item.id}
                    onRefresh={this.handleRefresh}
                    refreshing={this.state.refreshing}
                    renderItem={ ({item}) =>
                        <TouchableOpacity style={styles.listitem}
                            onPress={ () => {
                                if (item.type == 'versions')
                                    mTabNav.navigate('ViewerTab', {urn: item.id, ref
                                else {
                                    navigate('Links', {headerTitle:item.name, selecte
                                }
                            }}>
                            <Text>{item.name}</Text>
                        </TouchableOpacity>

                    }
                />
            </View>
        );
    }
}

const StackNav = StackNavigator({
    Links: {
        screen: FileListNav,
        navigationOptions: {
            headerBackTitle: null
        },
    }
}, {
    initialRouteParams: {
        headerTitle: 'Hubs',
        type: 'hubs',
        refreshing: false
    }
});

export default class FileList extends React.Component {
    render() {
        mTabNav = this.props.navigation;
        return <StackNav / >
    }
}
```

AUTODESK UNIVERSITY

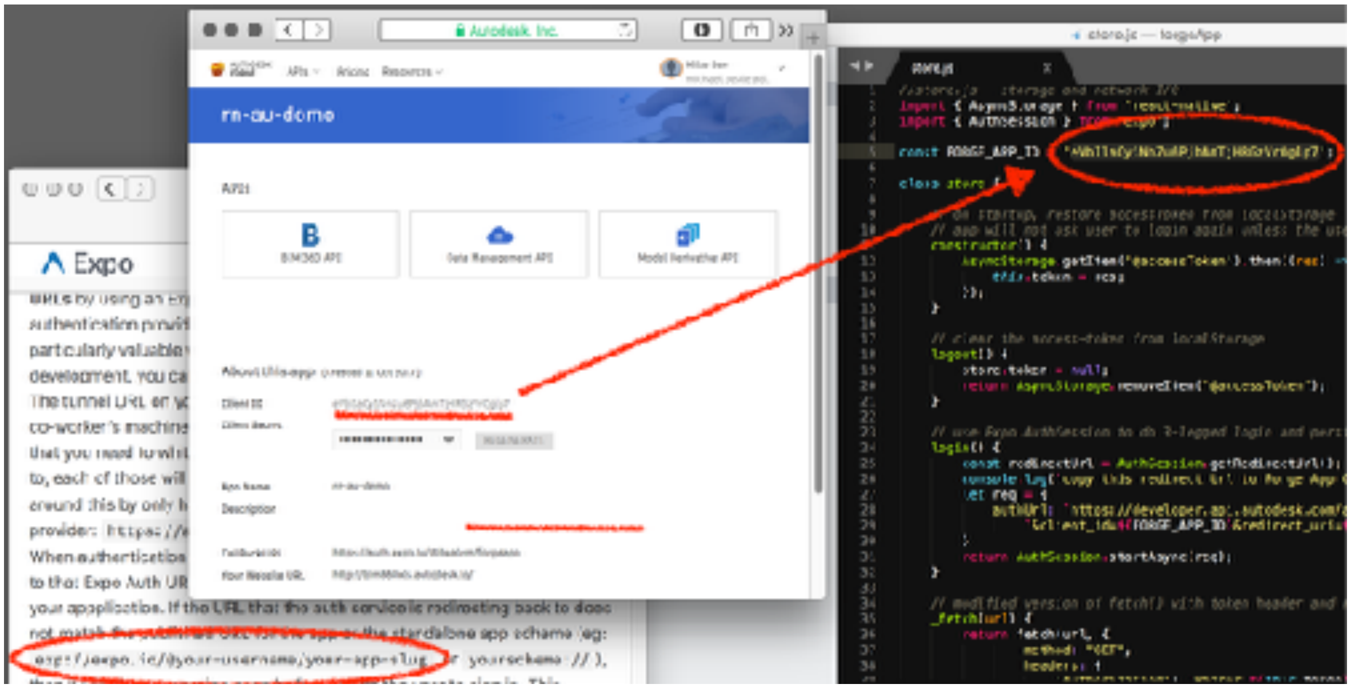
3. Adding 'Login' and Forge Data

So far, we have created a 'shell' app which has dummy data. Now the tricky part - Let's replace the dummy data with real data from the Forge Services.
To do that, we need to update **FileList.js , viewer.js** and add a new file **Store.js**

Setup

1 Go ahead and copy these files from GitHub here:
2 Create a new Forge App (follow this tutorial)
3 copy your new Forge App's **CLIENTID** into the variable **'FORGE_APP_ID'** in file **'store.js'**
4 In the Forge Callback, set the 'Redirect URL' to this...
5 https://auth.expo.io/@your-username/your-app-slug

> where username and app are based on your Expo account and App.
Here is a screenshot of these changes:



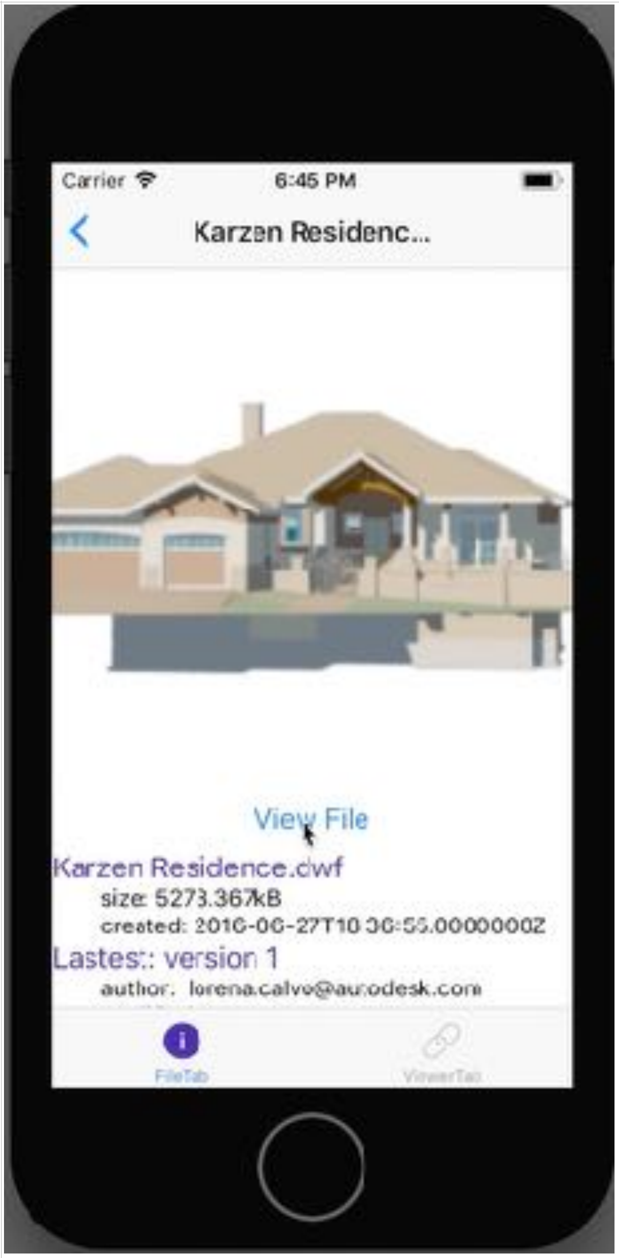
```
//store.js - storage and network I/O
import { AsyncStorage } from 'react-native';
import { AuthSession } from 'expo';

const FORGE_APP_ID = '[illegible]';

class store {

    // on startup, restore accessToken from localStorage
    // app will not ask user to login again unless the us
    constructor() {
        AsyncStorage.getItem('@accessToken').then((res) =>
            this.token = res;
        });
    }

    // clear the access-token from localStorage
    logout() {
        store.token = null;
        return AsyncStorage.removeItem('@accessToken');
    }

    // use Expo AuthSession to do 3-legged login and pers
    login() {
        const redirectUrl = AuthSession.getRedirectUrl();
        console.log(`copy this redirect Url to Forge App
        let req = {
            authUrl: `https://developer.api.autodesk.com/
                `&client_id=${FORGE_APP_ID}&redirect_uri=
        }
        return AuthSession.startAsync(req);
    }

    // modified version of fetch() with token header and
    _fetch(url) {
        return fetch(url, {
                method: "GET",
                headers: {
```

exp://expo.io/@your-username/your-app-slug

Setup of Callback-URL and Client-ID

1. Now refresh your Phone and the app will launch
2. click the ‘Login’ button on your iPhone
3. Use your Autodesk credentials to login, allow Access.

If everything works, you should see...



Carrier 6:45 PM

Karzen Residenc...

View File

Karzen Residence.dwf

size: 5273.367kB

created: 2016-06-27T10:36:55.0000000Z

Lastes:: version 1

author: lorena.calvo@autodesk.com

> Scan QR Code

… the Autodesk login box appears, you type in your Forge login credentials, allow access, and the app refreshes the file list. You are logged in and ready to navigate through your files and folders.

So what has changed?

FileList.js

1. The static list in fileList.js has been replaced with calls to store.js
2. The method 'handleRefresh' processes ForgeDM data into folder navigation
3. We added a login and logout button that uses AuthSession library from Expo
4. We added a spinner during loading ('pull to refresh' UI)
5. We added a 'file details' UI

Digging into the Store.js file

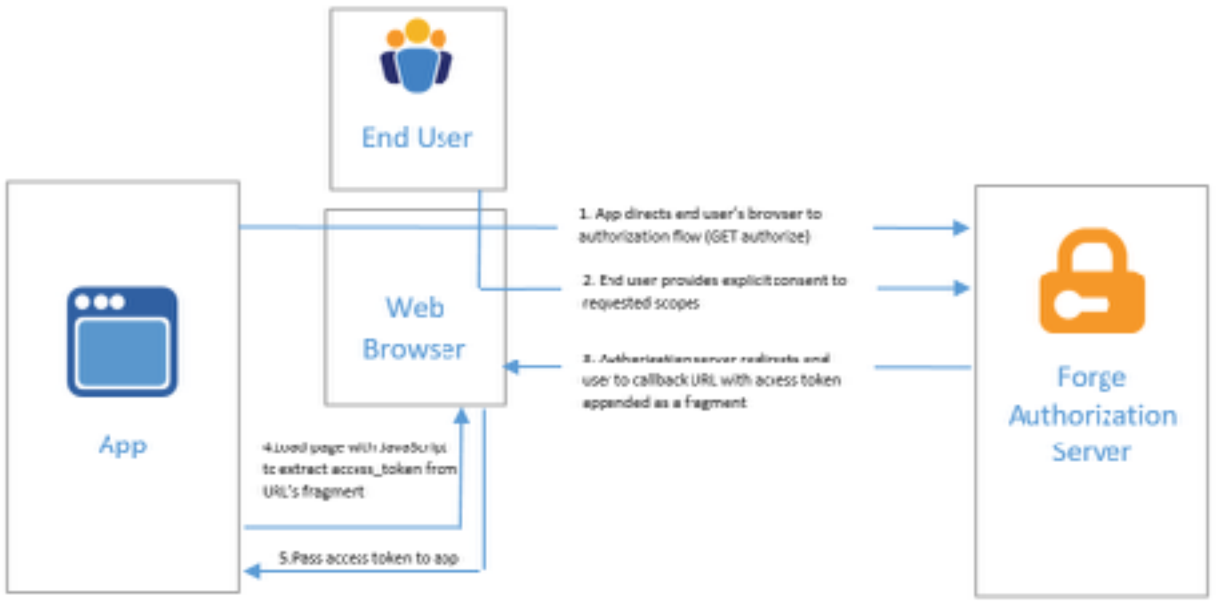
1. store is a singleton class (line 90)
2. it keeps application state (in future, I'll use MobX)
3. it handles network IO to the Forge API (line 69-86)
4. it remember's the user's login by persisting the accessToken in AsyncStore (line 11-14)

```
//store.js - storage and network I/O
import { AsyncStorage } from 'react-native';
import { AuthSession } from 'expo';

const FORGE_APP_ID = 'eVbIlaCy1NnZu6PjbAmTjHRGzVrOglp7';

class store {

// on startup, restore accessToken from localStorage
// app will not ask user to login again unless the user signs out.
constructor() {
AsyncStorage.getItem('@accessToken').then((res) => {
this.token = res;
});
}

// clear the access-token from localStorage
logout() {
store.token = null;
return AsyncStorage.removeItem('@accessToken');
}

// use Expo.AuthSession to do 3-legged login and persist resulting Access-Token
login() {
const redirectUrl = AuthSession.getRedirectUrl();
console.log(`copy this redirect Url to Forge App Callback: ${redirectUrl}`);
let req = {
authUrl: `https://developer.api.autodesk.com/authentication/v1/authorize?
response_type=token` +
`&client_id=${FORGE_APP_ID}&redirect_uri=${encodeURIComponent(redirectUrl)}
&scope=data:read`
}
return AuthSession.startAsync(req);
}

// modified version of fetch() with token header and response parsed as JSON
_fetch(url) {
return fetch(url, {
method: "GET",
headers: {
'Authorization': `Bearer ${this.token}`
}
})
.then(res => res.json())
.catch((err) => console.log(err));
}

// normalize the item for the UI
filterItem(i) {
return {
href: i.links.self.href,
type: i.attributes.extension.type,
name: i.attributes.name ? i.attributes.name : i.attributes.displayName,
id: i.id,
}
}

// converts Forge API json structure into a flat list of Items for the UI
filterList(res) {
// reached a leaf of the tree
if (!res.data)
return this.item(res.data);

// reached a leaf of the tree
return res.data.map((i) => {
return this.filterItem(i)
});
}

// retrieve the root list of hubs from Forge DM API
getHubs() {
return this._fetch('https://developer.api.autodesk.com/project/v1/hubs');
}

// retrieve child list from Forge DM API, based on the type
// where type = '/projects', '/topFolders', '/contents', '/tip'
getBranch(item) {
let ext = '';
if (item.type == 'folders:autodesk.core:Folder') {
ext = '/contents';
} else if (item.type == 'projects:autodesk.core:Project') {
ext = '/topFolders';
} else if (item.type == 'hubs:autodesk.a360:PersonalHub') {
ext = '/projects';
} else if (item.type == 'items:autodesk.core:File') {
ext = '/tip';
}
return this._fetch(`${item.href}${ext}`);
}
}
// store is a singleton, instance is created on startup
module.exports = new store();
```

Explaining 3-legged OAuth on mobile:

During the last part of the 3-legged oAuth handshake (see diagram, step 3,4 & 5), our mobile app needs to get a callback from the Forge API. Obviously, configuring the Forge callback to point to a mobile phone's IP address directly is a "bad idea", so we need to use a 'middle man' (3rd party cloud server) that can associate a mobile device with a Forge callback.

Currently I use Expo service with the AuthSession component.


End User
App
Web Browser
Forge Authorization Server
1. App directs end user's browser to authorization flow (GET authorize)
2. End user provides explicit consent to requested scopes
3. Authorization server redirects end user to callback URL with access token appended as a fragment
4.Load page with JavaScript to extract access_token from URL's fragment
5.Pass access token to app

Our sample app requests an access-token via an 'Implicit Grant' directly to the Forge Service. You can see this in the file store.js/login() line 24-32. The url also contains a special callbackURL, that goes back to the Expo server - our 'middle man'.

The Expo Server, has a 'session cookie' of each iPhone/Android's IP address and transmits the access-token message. AuthSession library is waiting for this message from the Expo server and finally passes it to our React Native app (that's a promise, excuse the pun) and we then use it in all future 'fetch' requests to the Forge APIs.

In the original NodeJS sample, we hosted a Heroku nodeJS server to connect the browser to the Forge callback, using a session cookie. Before we publish our app to the Apple/Android App Store, we should consider replacing the Expo service with our own server. We will follow up in a later tutorial using Serverless (see my previous Blog: Serverless and ClaudiaJS).

4. Sharing your App with other iPhone users

One of the reasons for using Expo and XDE was it's feature to 'Publish your app'. This gives you a *very* convenient way to let others try out your new app on their own phone. It delays the decision to buy an Apple Developer Account, MacBook, install Xcode, build and deploy. Just share the QRCode, like the one I posted in the title of this blog, and away you go !
Remember to follow @micbeale on Twitter.
..Feel free to add any issues you find to my Github issues repo.
Stay tuned for part Two !
----

5. Headless Viewer

Let’s take a closer look at the viewer tab. This is where the Forge Viewer is displayed in a webview.

Unfortunately, the black UI is made from HTML5 and, while it is passable for now, at some point we want to give our app a truly native UI with our own styling.

To do this, we are going to use a special version of the viewer that in ‘headless’. It has all of the UI code removed, making the viewer smaller and canvas rendering slightly more efficient.

We just need to change two things in our code.

First, we swap … // viewer3D.min.js

with …. firefly.min.js

and secondly, we will replace GuiViewer3D with Viewer3D, like this:

```
// viewer = new Autodesk.Viewing.Private.GuiViewer3D(viewerDiv);
viewer = new Autodesk.Viewing.Viewer3D(viewerDiv, options);
```

> Sample code can be found on github account (github.com/wallabyway/forgeApp)

Now, if we run the app, we will see the viewer with no UI.

Time to add back some custom UI. Let’s add a simple section toolbar to get started.

6. Custom Toolbar

This is the beginning of adding your own custom UI. We will get started with a simple toolbar UI. The toolbar will only expose the sectioning command found in the Forge Viewer API.

Here is the updated viewer.js file containing the new native UI, and the javascript bridge code to tell the viewer how to section the drawing.

```
class TabIcon extends Component {
  render() {
    return (
        <TouchableOpacity onPress={this.handlePress.bind(this)}>
        <Text style={{backgroundColor:(this.props.active) ? '#000' : '#0000', color:'white',
          padding:15, margin:0, paddingTop:15, paddingBottom:10, fontSize: 28,
          fontFamily: 'fontello'}}>{this.props.char}</Text>
        </TouchableOpacity>
    );
  }
  handlePress(e) {
    if (this.props.onPress) {
      this.props.onPress(e);
    }
  }
}
```

```
const sectionToolbar = (
  <View style={styles.toolbar}>
    <Slider style={styles.slider} minimumValue={-8} maximumValue={5}
          onValueChange={(value) => onSectionSliderChange(value)} />
    <TabIcon char='t' active={this.state.sectionDirection === 0} onPress={() =>
          {this.setState({sectionDirection: 0, szSectionPlane:'0,1,0'}) }}/>
    <TabIcon char='u' active={this.state.sectionDirection === 2} onPress={() =>
          {this.setState({sectionDirection: 2, szSectionPlane:'0,0,1'}) }}/>
    <TabIcon char='s' active={this.state.sectionDirection === 1} onPress={() =>
          {this.setState({sectionDirection: 1, szSectionPlane:'1,0,0'}) }}/>
  </View>
)
```

7. Offline Files

The last part of this code is to explain one technique for offline viewing that leverages the cache.manifest approach. We will create a simple cache.manifest file, and have the viewer point to it.

Unfortunately, it is not a final solution. Instead, I recommend switching to the higher performance wkWebView browser component, I'll discuss in the 'future components' section next.

See here for more details: https://forge.autodesk.com/blog/viewer-airplane-mode

cache.manifest file for the shaver files (which I uploaded a fusion file and then downloaded the assets using the forge gallery site: https://forge-rcdb.autodesk.io/gallery)

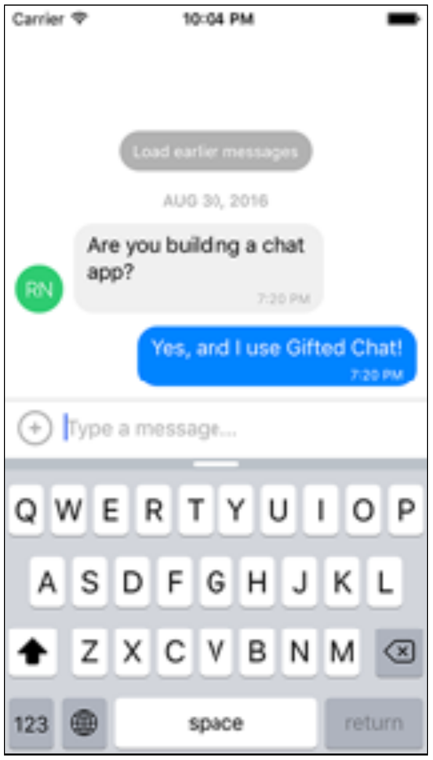
```
CACHE MANIFEST

https://developer.api.autodesk.com/viewingservice/v1/viewers/three.min.js
https://developer.api.autodesk.com/viewingservice/v1/viewers/viewer3D.min.js
https://developer.api.autodesk.com/viewingservice/v1/viewers/style.min.css

https://developer.api.autodesk.com/viewingservice/v1/viewers/wgs.js
https://developer.api.autodesk.com/viewingservice/v1/viewers/lmvworker.min.js
https://raas-assets.autodesk.com/StaticContent/BaseAddress
https://raas-assets.autodesk.com/StaticContent/BaseAddress
https://ase-cdn.autodesk.com/adp/v1.0.3/js/adp-web-analytics-sdk.min.js

shaver/0.svf
shaver/geometry0.pack
shaver/geometry1.pack
shaver/geometry2.pack
shaver/geometry3.pack
shaver/image0.png
shaver/objects_attrs.json.gz
shaver/objects_avs.json.gz
shaver/objects_ids.json.gz
shaver/objects_offs.json.gz
shaver/objects_vals.json.gz

NETWORK:
*
```

Future Componts

1. messaging (gifted chat)

https://github.com/FaridSafi/react-native-gifted-chat

```
import { GiftedChat } from 'react-native-gifted-chat';

class Example extends React.Component {

  state = {
    messages: [],
  };

  componentWillMount() {
    this.setState({
      messages: [
        {
          _id: 1,
          text: 'Hello developer',
          createdAt: new Date(),
          user: {
            _id: 2,
            name: 'React Native',
            avatar: 'https://facebook.github.io/react/img/logo_og.png',
          },
        },
      ],
    });
  }

  onSend(messages = []) {
    this.setState((previousState) => ({
      messages: GiftedChat.append(previousState.messages, messages),
    }));
  }

  render() {
    return (
      <GiftedChat
        messages={this.state.messages}
        onSend={(messages) => this.onSend(messages)}
        user={{
          _id: 1,
        }}
      />
    );
  }

}
```



Are you buildng a chat app?
7:20 PM
Yes, and I use Gifted Chat!
7:20 PM

2. taking photos with camera

https://github.com/lwansbrough/react-native-camera

3. native notifications

https://docs.expo.io/versions/latest/guides/push-notifications.html

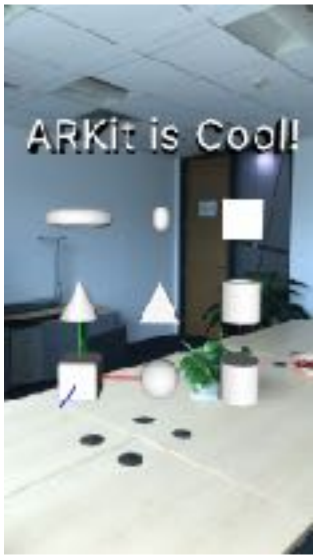
4. maps (mapbox)

https://github.com/airbnb/react-native-maps

5. demo of AR markup with react-native

see 3Dmarkup demo at vrock.it website

https://github.com/HippoAR/react-native-arkit



Conclusion

We've covered a lot of material and you hopefully have a better understanding of what React-Native mobile development looks like and are excited about building your next Forge mobile app with React Native. I can't wait to see what you come up with !

You can follow me on the Forge blog post here: https://forge.autodesk.com/blog/forge-react-native-au-talk
And you can also follow me on twitter here: twitter.com/micbeale

Stay tuned for my part Two of my React-Native blog post !

Resources:

1. Source Code: github.com/wallabyway/forgeApp

2. To setup React Native for debugging, install VSCode and follow this youtube example

3*. To speed up development, activate Hot Reloading:* Shake the phone and tap 'disable live reload', and then tap 'enable hot-reload'