
Reaching React Native's limits
How to use native code in React Native
by Simon Auer

About me

- Du Digital Solutions
  (React, React native, React360, Graphql, ...)
- <3 macaroni and cheese
  instagram.com/kasnudelmccheese
- React ~5 years
  React Native ~3 years

E: simon@du.digital

G: github.com/SimonErich

L: www.linkedin.com/in/simon-auer/

T: twitter.com/SimonEritsch

What is React Native

- Develop apps for Android & iOS (& Windows)
- Uses similar syntax to React
- Allows you to use tools already know (JS, Html like tags and CSS)

React Native vs Hybrid apps (Cordova/Ionic/...)

Hybrid Apps	React-Native
HTML, JS, CSS Application in Webview (basically just a browser without address bar)	Native UI building blocks (coordinated by Javascript)
Normal HTML Tags (div, p, span, a)	HTML like tags (View, Text, Button)
Limited access to native parts	Access to native parts via bridge

Live coding - step #5

Install your newly created module:

We have now created a module, that we could even publish on github or via npm.
Because we don't want to do that right now, we have placed it within our "modules" folder for now.

React Native requires our modules to be in the "node_modules" folder by default.
So, let's install our modules via npm.

To be able to install, not from a published npm repository, but from our "modules" directory, we will use the value "file:..." instead of the normal version number in our package.json file.

So go ahead now and open the app root package.json file and add the following dependency:

```
{
  ...
  "dependencies": {
    ...
    "react-native-sandwich-maker": "file:./modules/SandwichMaker"
  },
  ...
}
```

And then run yarn/npm install in your terminal to install the new dependency:

```
$ npm install
```

Bridging in React Native

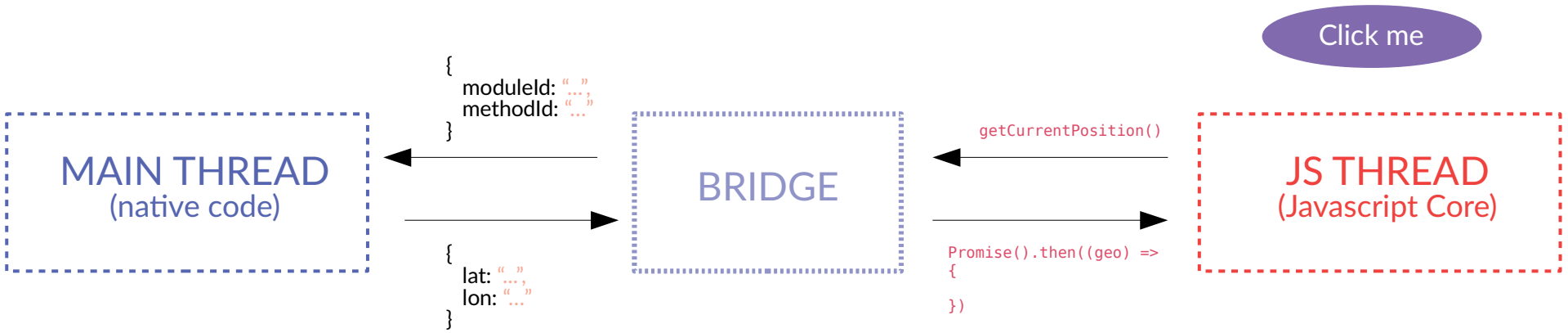
logic for rendering and bootstrapping

Communication between MAIN and JS thread

- async
- batched

Main app logic in javascript

Bridging example



2 types of native modules in React Native

Native Modules	UI Components
• Not visible • Mostly APIs, Data driven • Analatics, Sensors, Beacons, ...	• Visible • User Interface blocks • Video player, Ad Banner, ...

Live coding - step #1

Install react-native-cli and react-native-create-library:

First we need to install the react-native-cli and react-native-create-library to generate our module boilerplate later on.
Use the following commands in your terminal:

```
$ npm install -g react-native-cli
$  npm install -g react-native-create-library
```

Create a new React Native project:

Now we are going to create our first react-native project “mydemo” using the react-native-cli.

```
$ react-native init mydemo
$ cd mydemo
```

Initialize the project:

Now launch Android Studio first and then open the “android” folder as a project in your project directory.
This will initialize the project and install the first required gradle dependencies and link the SDK correctly.
(If you have never used Android Studio before, you might have to install the SDKs)

Now start an emulator using Android Studio. (tutorials on how to setup and start emulators in Android Studio are available everywhere out there)

Live coding - step #2

Start the application:

If that works, go ahead and launch your application for the first time.
To do this open your terminal again, go to your project folder root and start the bundler (takes care of bundling the javascript part of your application):

```
$ yarn start
```

Now with the bundler running, we can start the application on our emulator by calling in new tab. (the old tab needs to stay open and running the bundler).

```
$ react-native run-android
```

Create your native module:

Now that we know our application works, (hopefully) we are going to create our first native module.
We will first create a directory within our project root to keep the package repository:

```
$ mkdir modules
$ cd modules
```

And now we will create the module, using react-native-create-library

```
$ react-native-create-library SandwichMaker
```

Live coding - step #3

Add a simple method to the native module code:

Using native code is very simple in React Native.

1.) Create or find a native functionality you want to use in React Native
For example: We want to use the native “in app” notification system of Android (Toasts)
To show a specific message.

2.) To implement this functionality, we need to create a simple function that wraps this functionality and add it to our *modules/SandwichMaker/android/src/main/java/com/reactlibrary/RNSandwichMakerModule.java file.*

```
@ReactMethod
public void show(String message, int duration) {
    Toast.makeText(getReactApplicationContext(), message, duration).show();
}
```

3.) Also add the import statement for “Toast”, to have Android’s Toast methods available.

```
import android.widget.Toast;
```

You can write as many functions as you want. Notice the **@ReactMethod** annotation! This annotation tells React Native, that this method should be usable from within the JS thread/your JS code. If you want to export a function to be usable by JS, always annotate your custom methods with this one and your good to go.

Live coding - step #4

Install your newly created module:

We have now created a module, that we could even publish on github or via npm.
Because we don't want to do that right now, we have placed it within our "modules" folder for now.

React Native requires our modules to be in the "node_modules" folder by default.
So, let's install our modules via npm.

To be able to install, not from a published npm repository, but from our "modules" directory, we will use the value "file:..." instead of the normal version number in our package.json file.

So go ahead now and open the app root package.json file and add the following dependency:

```
{
  ...
  "dependencies": {
    ...
    "react-native-sandwich-maker": "file:./modules/SandwichMaker"
  },
  ...
}
```

And then run yarn/npm install in your terminal to install the new dependency:

```
$ npm install
```

Now we have our module installed in node_modules. We just need to link it.
(linking will add all the necessary native bootstrapping code in our MainApplication.java (Android) and add the module pod (iOS).

```
$ react-native link react-native-sandwich-maker
```

Live coding - step #5

Using our newly created module:

Now that we have created a native “export” method in our module, our library, created by react-native-create-library, will automatically export this method for use in javascript within the modules/SandwichMaker/index.js file.

So let’s just use this code in our project. Open the **App.js** file and add the following imports:

```
import RNSandwichMaker from 'react-native-sandwich-maker';
import { Button } from 'react-native';
```

And then add the following Code just below the last <Text ...> tag:

```
<Button onPress={() => { RNSandwichMaker.show('Hello beautiful world', 3000); }}
title="Click me" />
```

Live coding - step #6

Restart the application and see the awesomeness:

Alright. Everything should be setup and hopefully work now.
Let's restart our app from our terminal (make sure your emulator is still open):

```
$ react-native run-android
```

I hope everything went well and works. If you have any questions or problems just let me know in the React Vienna slack channel.
(https://reactvienna.slack.com)

I will try to look in there more often at the moment. Just write it in the general channel and ping me, then I or somebody else will try to help out.

Good luck =)

Live coding - step #7

What about iOS

For the sake of brevity, I have just included an example for Android here. To be able to use this with iOS, everything is exactly the same, apart from “Live coding – step #3”, which will be iOS specific.

For iOS you don’t have to open the of *RNSandwichMakerModule.java* course, but the modules/SandwichMaker/ios/RNSandwichMaker.m file and add your method in the following format after RCT_EXPORT_MODULE();

```
RCT_EXPORT_METHOD(addEvent:(NSString *)message)
{
  RCTLogInfo(@"Hello message: %@", message);
}
```

You also have to add the import here for RCTLogInfo.
(Note this is not exactly the same as in Android, because there are no toasts on iOS and this is just a simple example of logging)

```
#import <React/RCTLog.h>
```

Apart from that, everything should be exactly the same. :)

Further links and information

Android:

https://facebook.github.io/react-native/docs/native-modules-android (without UI)

https://facebook.github.io/react-native/docs/native-components-android (with UI)

iOS:

https://facebook.github.io/react-native/docs/native-modules-ios (without UI)

https://facebook.github.io/react-native/docs/native-components-ios (with UI)

Pitfalls and tips:

- Limit passes on messagequeue
- Use debugging tools (Native debuggers/profilers, rn-snoopy, …)
- Keep data flat and lightweight (no files, …)
- Keep complex business logic in JS or Main
- Use “usenative” flag for animations

Thank you

E: simon@du.digital

G: github.com/SimonErich

L: www.linkedin.com/in/simon-auer/

T: twitter.com/SimonEritsch

We are hiring:

ReactJS / React Native devs:

du.digital/jobs.html