React Native has grown to be one of the most popular platforms for building native apps, being used by companies like Tesla, Instagram, and Facebook in production. React Native allows you to write apps in JavaScript that are rendered with native code. It has many of the features that you value when working with Meteor, like instant refresh on save.

You can easily integrate your React Native app with Meteor, using the same methods you would on a Meteor + React Web app. The integration supports most Meteor features, including Methods, Pub/Sub, and Password Accounts, and has the same usage as `react-meteor-data`.

Getting started with React Native

React Native projects are coded using the same React principles, but have a completely separate codebase from your Meteor project.

A collection of NPM packages are being developed to make it easy for you to integrate React Native with Meteor. In order to use React Native with Meteor, you create a React Native app and use the `@meteorrn/core` package to connect your app to your Meteor server. The `@meteorrn/core` package contains Meteor, MongoDB, `withTracker`, Accounts, and more.

For most projects, since your native app will display the same data and call the same methods as your Meteor web app, creating a React Native app that connects to your Meteor server does not require any changes to your Meteor codebase.

The only time you will need to make changes to your Meteor codebase is to enable certain features that are unique to your native app. For example, if you want to add push notifications to your native app, you will need to create a method on your Meteor app to store the native push tokens for a user.

There are two routes for getting started with React Native. You can use "Vanilla" React Native, or you can use Expo. Expo is a set of tools built around React Native. You can even try out React Native from your web browser using Expo Snack. You don't even need to install XCode or Android Studio to start using Expo.

Here are the downsides to using Expo:

- You cannot add Native Modules that use Native Code (Java, Swift, etc)
- You cannot use packages that require linking (these are npm modules that include native code, and allow you to acess native features like the camera, push notifications, fingerprint authentication, etc). \
- Apps that use Expo are much larger then pure React Native apps

Expo does provide some native features (click here for the full list), but if there is a feature missing that you need, you'll likely need to use an npm package or your own custom native code.

You can "eject" your app from Expo to take advantage of Vanilla React Native features, but ejection cannot be undone easily.

The React Native documentation lets you choose between the Expo ("Expo CLI") and Vanilla React Native ("React Native CLI") setup instructions. You can read through the installation instructions and decide which option makes more sense for you.

Here is the link to the React Native getting started documentation: https://reactnative.dev/docs/environment-setup

Once you have your environment setup and have your app running on your device or in the emulator, you can proceed to the next step of the guide: "Meteor React Native Installation"

Meteor React Native Installation

To install the `@meteorrn/core` package, run the following command in your React Native project:

```
npm install --save @meteorrn/core
```

You also need to confirm you have the package's peer dependencies installed:

- Confirm you have `@react-native-community/netinfo` installed
- Confirm you have `@react-native-async-storage/async-storage@>=1.8.1` installed. If you are using Expo, or otherwise cannot use `@react-native-async-storage/async-storage`, please see these instructions.

The `@meteorrn/core` package enables your React Native app to establish a DDP connection with your Meteor server so it can receive data from publications and call server methods. It also provides access to core Meteor client methods like `Accounts.createUser` and `Meteor.loginWithPasword`, and allows you to display data in your app with the `withTracker` method.

Note: If your React Native app uses version 0.59 or lower, the @meteorrn/core package contains breaking changes. Use react-native-meteor instead.

Setup

First, import `Meteor`, `withTracker`, and `Mongo`:

```
import Meteor, { Mongo, withTracker } from '@meteorrn/core';
```

Next, you need to connect to your Meteor server. This should typically be at the start of your App.jsx.

```
Meteor.connect("wss://myapp.meteor.com/websocket");
```

Define your collections:

```
const Todos = new Mongo.Collection("todos");
```

And now you're ready to start coding.

Coding with Meteor React Native

If you've used React before, coding with React Native is pretty straightforward. However, instead of components like `div` and `span`, we have `View` and `Text`. You can learn the fundamentals of React Native here.

Meteor React Native's usage is designed to be as close to `meteor/react-meteor-data` and the Meteor core as possible. It provides a `withTracker` method. The package also has full support for accounts, including `Meteor.loginWithPassword`, `Meteor.user`, `Accounts.createUser`, `Meteor.loggingIn`, `Accounts.forgotPassword`, etc.

```
const MyAppContainer = withTracker(() => {

    const myTodoTasks = Todos.find({completed:false}).fetch();
    const handle = Meteor.subscribe("myTodos");

    return {
        myTodoTasks,
        loading:!handle.ready()
    };

})(MyApp);
```

When rendering small amounts of data, you can use the array map method:

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

class MyApp extends React.Component {
    render() {
        const { loading, myTodoTasks } = this.props;

        if(loading) {
            return <View><Text>Loading your tasks...</Text></View>
        }

        return (
            <ScrollView>
                {!myTodoTasks.length ?
                    <Text>You don't have any tasks</Text>
                :
                    myTodoTasks.map(task => (
                        <Text>{task.text}</Text>
                    ))
                }
            </ScrollView>
        );
    }
}
```

If you are rendering a large amounts of data, you should use the FlatList component.

Conclusion

Here are some useful links for futher reading:

You can see a list of example components built with `MeteorRN` here.

You can view the full API docs for `MeteorRN` on the meteor-react-native repo

You can see the official React Native API docs here

"How to setup your first app" from HackerNoon

"The Full React Native Layout Cheat Sheet" from WixEngineering