Пишем первое приложение на React Native. Часть 1.

Andrey Melikhov

Сообщество DevSchacht растёт, накапливает контент и завоёвывает преданных читателей. В свою очередь последние всегда желают получать информацию быстро и удобно. А лучше сразу на своё мобильное устройство. Можно сделать сайт, а можно пойти дальше и создать полноценное приложение, которое будет уведомлять о новых статьях, кешировать их и позволит читать там, где нет мобильной связи (например, в метро).

И тут у нас есть два варианта: новый кленовый PWA и старое доброе нативное приложение. К сожалению, PWA пока не работают полноценно на iOS (как минимум до появления сервис-воркеров в Safari). А написание нативных приложений требует знаний незнакомых нам технологий. Но мы фронтендеры и не хотим учить Java и Swift. Мы хотим React, CSS и в продакшен, и благодаря компании Facebook у нас есть всё, для того, чтобы реализовать наше желание.

В этой серии статей мы постараемся шаг за шагом разработать мобильное приложение DevSchacht на React Native и довести его до публикации.

Дисклеймер: автор не является профессиональным разработчиком на React Native. Советы, данные в статье, могут быть не оптимальны — но, как говорится, мы открыты для пул-реквестов :)

Что такое React Native?

React Native — это фреймворк для разработки кроссплатформенных приложений. Он даёт возможность создавать и использовать компоненты точно так же, как обычно мы это делаем в React, вот только рендериться они будут не в HTML, а в нативные контролы операционной системы, под которую будет собрано наше приложение.

Итак, у нас есть знакомый JavaScript, JSX и CSS (на самом деле это полифил, реализующий подмножество CSS). С JavaScript есть некоторая неприятная особенность: и на Android, и на iOS ваш код будет исполнять движок JavaScriptCore (тот самый, который идёт в комплекте с WebKit). А вот отлаживать код в режиме дебаггера и запускать тесты вы будете в node.js и Chrome, то есть на движке V8.

Запомните:

- На симуляторах и устройствах iOS, эмуляторах и устройствах Android React Native использует JavaScriptCore. В iOS JavaScriptCore не использует JIT.
- При использовании дебаггера весь код запускается внутри среды отладки (например, в Chrome), общение с нативным кодом происходит через

WebSocket. Таким образом, при отладке вы используете V8.

Инструменты

Среда разработки

Сам я являюсь ярым поклонником WebStorm, это немного тяжеловесная и интерфейсно перегруженная, но потрясающе мощная и удобная IDE на основе IntelliJ IDEA от компании JetBrains. В настоящее время в неё неплохо интегрирован набор утилит для работы с React и React Native.

Когда возможностей WebStorm становится слишком много, а свободного ОЗУ в системе — слишком мало, я расчехляю старый добрый vim. В принципе, никто не запрещает использовать его с React Native, особенно если подключить подсветку JSX-синтаксиса.

Так же, можно дать шанс результату симбиоза Facebook и GitHub — редактору Nuclide. Этот редактор является набором расширений для Atom и позиционируется компанией Facebook как первоклассное решение для разработки на React Native. Честно говоря, в моём случае этот редактор оказался невероятно требователен к ресурсам и я отказался от его использования.

Заготовка

Для разработки нашего приложения воспользуемся заготовкой от Facebook — Create React Native App. Установить её не сложно:

```
$ npm install -g create-react-native-app
```

К сожалению, если вы уже установили пятую версию npm, то ничего не получится. Либо ставьте четвёртую версию npm, либо попробуйте yarn — новый пакетный менеджер от Facebook, в котором всё отлично работает (что неудивительно).

Далее создаём каркас нашего будущего приложения:

```
$ create-react-native-app devschacht
```

Тулинг

В принципе, у нас уже есть всё для начала разработки. Однако стоит установить ещё пару полезных утилит, которые упростят нам жизнь в будущем.

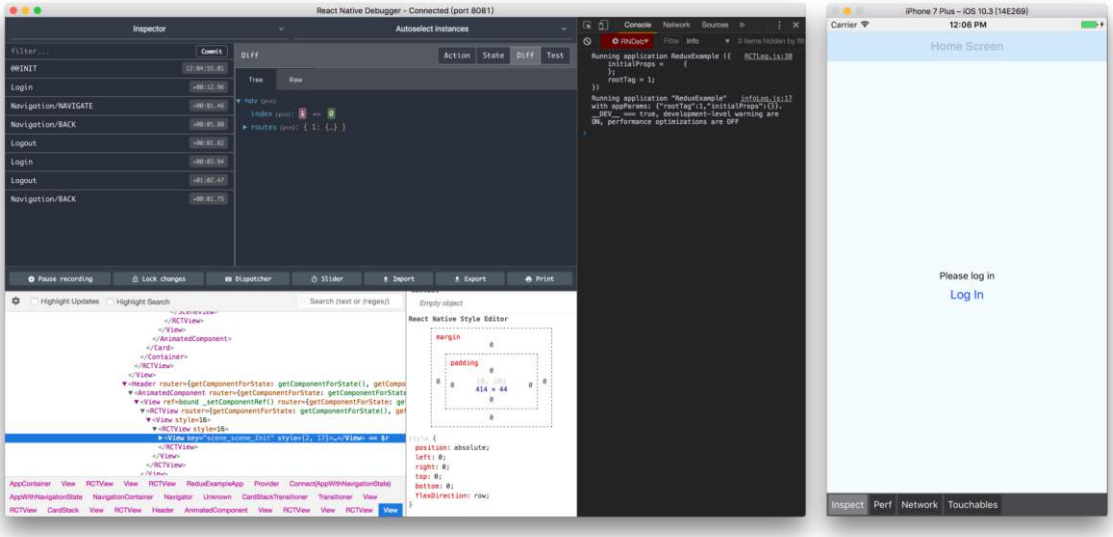
Create React Native App поставляется вместе с Expo. Expo — это набор утилит, библиотек и сервисов, облегчающих разработку на React Native. Expo SDK позволяет обращаться к системной функциональности (такой как камера, контакты, локальное хранилище данных и так далее). Это значит, что вам не нужны Xcode или Android Studio и умение писать нативый код. А так же это значит, что благодаря этому слою абстракции, ваш код становится действительно кроссплатформенным.

Более того, вам даже не нужен XCode и симулятор iOS для запуска приложения, с помощью Expo приложение в режиме отладки можно запустить прямо на телефоне. Для этого на телефон нужно установить клиент Expo для iOS или Android.

Так же, для большего удобства разработки, рекомендую установить Expo XDE, существующий в версиях под MacOS, Windows и Linux.

And last but not the least (*последний, но тоже важный*): нам нужен хороший дебаггер. По умолчанию в качестве дебаггера открывается Google Chrome. Это неплохо, но недостаточно

удобно. Есть сторонние дебаггеры, один из лучших это React Native Debugger.



Так как стандартный упаковщик (или packager — утилита, упаковывающая ваш JavaScript код в бандл) React Native запускается на порту 8081, а упаковщик Expo на порту 19001, то в дебаггере нужно указать этот порт.

Привет, мир!

Итак, всё готово, для того, чтобы попробовать запустить наше первое приложение. Нам надо выбрать, где мы запустим наше приложение. Так как я работаю на MacOS и у меня установлен XCode, то я выбираю симулятор.

```
$ cd devschacht
$ npm run ios
```

Если у вас нет MacOS, то вы можете запустить приложение прямо на устройстве. Откройте клиент Expo и просканируйте QR-код, который выведется после запуска.

```
$ cd devschacht
$ npm start
```

Вы должны увидеть на экране следующий текст

```
Open up App.js to start working on your app!
Changes you make will automatically reload.
Shake your phone to open the developer menu.
```

Открываем App.js и меняем

```
<Text>Open up App.js to start working on your app!</Text>
<Text>Changes you make will automatically reload.</Text>
<Text>Shake your phone to open the developer menu.</Text>
```

на

```
<Text>Привет, мир!</Text>
```

Сработает Hot Reload (горячая перезагрузка), и в приложении вы увидите «Привет, мир!».

Что тут происходит? Наш код шаблона выглядит как обычный HTML, но вместо веб-элементов, таких как `<div>` или `<span>`, мы используем React-компоненты. В данном случае `<Text>` является встроенным компонентом, отвечающим за отображение текста.

```
export default class App extends React.Component {
  render() {
    return (
      <View style={styles.container}>
        <Text>Привет, мир!</Text>
      </View>
    );
  }
}
```

Этот код определяет новый компонент `App`. Все, что вы видите на экране - это набор компонентов. Компонент может быть довольно простым: единственное, что требуется, это функция рендеринга, возвращающая JSX. В следующий раз я покажу как можно писать компоненты в более компактном стиле.