8

Speak React Native

React Native course by U+_

Overview

- Testing
- Unit tests
- Snapshot tests
- End to end tests
- Translations

Project without Expo

Project without Expo

- New skeleton

  https://github.com/mamartin/srn-without-expo

- Run project

```
$ react-native run-android

$ react-native run-ios
```

Testing

Motivation

- Tests help us define what the code should do
- Writing tests prevents introducing new bugs when making changes to existing code
- Costs a little more time in the beginning, saves time long-term

Tools

- Jest / mocha
- React native test renderer
- Detox

Unit tests

- Best for simple functions which return simple values
- We define expected result for given arguments and check if a function returns it
- We **want** it to fail in case a change in implementation changes the result

Unit tests - a function to test

```
// utils/formatPrice.js
const formatPrice = (value: number | string) => {
  const numericValue = typeof value === 'string' ? parseInt(value, 10) :
value
  return isNaN(numericValue) ? '-' : `${numericValue.toFixed(2)}`
}

export default formatPrice
```

Our first unit test

```
// utils/__tests__/formatPrice.js
import formatPrice from '../formatPrice'

it('return formatted price', () => {
  const formattedPrice = formatPrice(3)
  expect(formattedPrice).toEqual('$3.00')
})
```

Our first unit test

```
// utils/__tests__/formatPrice.js
import formatPrice from '../formatPrice'

it('return formatted price', () => {
  const formattedPrice = formatPrice(3)
  expect(formattedPrice).toEqual('$3.00')
})
```

More tests of one function

```
it('return formatted price if value has a decimal point', () => {
 const formattedPrice = formatPrice(3.1)
 expect(formattedPrice).toEqual('$3.10')
})

it('return correct formatted price if value is string', () => {
 const formattedPrice = formatPrice('3')
 expect(formattedPrice).toEqual('$3.00')
})
```

Snapshot tests

- Compares result of a function call with previous result
- Can be used to test anything that can be converted to text / JSON
- Advantages - less manual work, makes us aware of changes when testing nested components

Snapshot test vs. unit test

- Both compare function result with given value
- **Unit test** - we define the value directly in the test and compare the result with it
- **Snapshot** - we save the value into a snapshot when we know that the function runs correctly, then compare with snapshot on subsequent runs

What is a snapshot?

- Physical file in the project structure (JSON format in our case)
- Generated, we don’t make manual changes to it
- We store the file in Git, so that everyone runs their tests against the same snapshot

Snapshotting components

```
import React from 'react'
import renderer from 'react-test-renderer'
import RoundedButton from '../RoundedButton'

test('renders correctly', () => {
 const tree = renderer
   .create(<RoundedButton onPress={() => null}>Ahoj</RoundedButton>)
   .toJSON()
 expect(tree).toMatchSnapshot()
})
```

Snapshotting components

```
import React from 'react'
import renderer from 'react-test-renderer'
import RoundedButton from '../RoundedButton'

test('renders correctly', () => {
 const tree = renderer
   .create(<RoundedButton onPress={() => null}>Ahoj</RoundedButton>)
   .toJSON()
 expect(tree).toMatchSnapshot()
})
```

…Generates something like this

```
// Jest Snapshot v1, https://goo.gl/fbAQLP

exports[`renders correctly 1`] = `
<View
  accessible={true}
  isTVSelectable={true}
  onResponderGrant={[Function]}
  onResponderMove={[Function]}
  style={
    Object {
      "alignItems": "center",
      "backgroundColor": "#ffffff",
      "borderColor": "rgb(24,202,167)",
      ...
```

Snapshotting everything

- Snapshotting is not only for components - we can make snapshots of other things, like JS objects
- For example redux actions - since action creators are functions returning objects, we can snapshot test them

Snapshotting everything

```
it('onGetMovies will generate action', () => {
  const action = JSON.stringify(onGetMovies())
  expect(action).toMatchSnapshot()
})

exports[`onGetMovies will generate action 1`] =
`"{\\"type\\":\\"ON_GET_MOVIES\\"}"`
```

Running tests

- Run unit & snapshot tests

```
yarn test
```

- Run tests & update snapshots

```
yarn test -u
```

Detox

Detox

- Automation framework for testing React native apps
- End to end testing, like a real user clicking through the app
- Also possible to run in cloud

Detox - installation (iOS)

- homebrew, node
- Install applesimutils

```
$ brew tap wix/brew

$ brew install applesimutils
```

A collection of utils for Apple simulators, Detox uses it to communicate with the simulator.

Detox - installation

```
$ npm install -g detox-cli
```

```
$ npm install detox --save-dev
```

```
$ npm install mocha --save-dev
```

Detox - set up (iOS)

- package.json

```
"detox": {
   "configurations": {
     "ios.sim.debug": {
       "binaryPath":
"ios/build/Build/Products/Debug-iphonesimulator/srnwithoutexpo.app",
       "build": "xcodebuild -project ios/srnwithoutexpo.xcodeproj -scheme
srnwithoutexpo -configuration Debug -sdk iphonesimulator -derivedDataPath ios/build",
       "type": "ios.simulator",
       "name": "iPhone 8"
     }
   }
```

Detox - set up (iOS)

- package.json

```
"detox": {
   "configurations": {
     "ios.sim.debug": {
       "binaryPath":
"ios/build/Build/Products/Debug-iphonesimulator/srnwithoutexpo.app",
       "build": "xcodebuild -project ios/srnwithoutexpo.xcodeproj -scheme
srnwithoutexpo -configuration Debug -sdk iphonesimulator -derivedDataPath ios/build",
       "type": "ios.simulator",
       "name": "iPhone 8"
     }
   }
```

Detox - writing test

- Elements must have test IDs

```
export default class RoundedButton extends React.PureComponent<Props> {
 render() {
   const { children, onPress} = this.props
   return (
     <TouchableOpacity onPress={onPress} style={styles.button}>
       <Text>{children}</Text>
     </TouchableOpacity>
   )
 }
}
```

Detox - writing test

- Elements must have test IDs

```
export default class RoundedButton extends React.PureComponent<Props> {
 render() {
   const { children, onPress, testID } = this.props
   return (
     <TouchableOpacity onPress={onPress} style={styles.button} testID={testID}>
       <Text>{children}</Text>
     </TouchableOpacity>
   )
 }
}
```

Detox - writing test

```
it("Should register user", async () => {
  await element(by.id("continueButton")).tap()
  await element(by.id("nameInput")).tap()
  await element(by.id("nameInput")).typeText("React Native")
  await element(by.id("finishButton")).tap() // hide keyboard
  await element(by.id("finishButton")).tap()
})
```

Detox - API examples

- Actions

```
await element(by.id('tappable')).longPress();
```

Detox - API examples

- Actions

```
await element(by.id('tappable')).longPress();
await element(by.id('tappable')).multiTap(3);
```

Detox - API examples

- Actions

```
await element(by.id('tappable')).longPress();

await element(by.id('tappable')).multiTap(3);

await element(by.id('scrollView')).swipe('down', 'fast', 0.5);
```

Detox - API examples

- Actions

```
await element(by.id('tappable')).longPress();
await element(by.id('tappable')).multiTap(3);
await element(by.id('scrollView')).swipe('down', 'fast', 0.5);
```

- Expectations

```
await expect(element(by.id('UniqueId'))).toBeVisible();     // 75 %
```

Detox - API examples

- Actions

```
await element(by.id('tappable')).longPress();
await element(by.id('tappable')).multiTap(3);
await element(by.id('scrollView')).swipe('down', 'fast', 0.5);
```

- Expectations

```
await expect(element(by.id('UniqueId'))).toBeVisible();     // 75 %
await expect(element(by.id('UniqueId'))).toExist();
```

Detox - run test

```
$ detox build

$ detox test
```

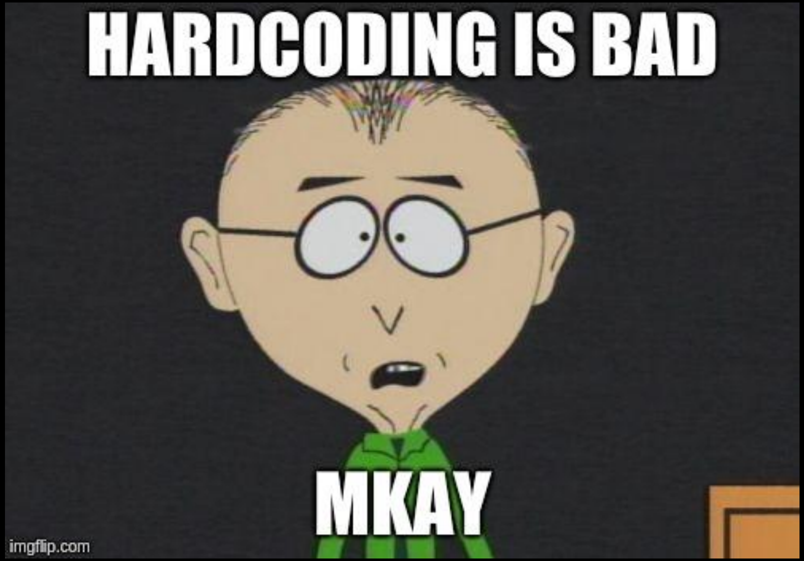
- Reusing existing build

```
$ detox test --reuse
```

Translations

Translations

- Hardcoding strings is bad, even if we have just one language
- JSON file with strings can easily be read and edited by anyone, not only devs



Translations

```
$ yarn add i18n-js
```

Translations

```
$ yarn add i18n-js
```

- i18n/en.js

```
export default {
 homeScreen: {
   title: "Welcome",
   description: "Some longer text",
 },
}
```

Translations

- Setting up

```
// containers/RootContainer.js
import en from "../i18n/en"
i18n.translations = { en }
```

Translations

- Setting up

```
// containers/RootContainer.js
import en from "../i18n/en"
i18n.translations = { en }
```

- Use

```
import i18n from "i18n-js"
<Text>{i18n.t("homeScreen.title")}</Text>
```

Translations - pluralization

```
// i18n/en.js
export default {
 homeScreen: {
   messages: {
     zero: "You have no messages.",
     one: "You have 1 message.",
     other: "You have {{count}} messages.",
   },
 },          // use in screen
}            <Text>{i18n.t("homeScreen.messages", { count: 3 })}</Text>
```

Translations - pluralization (CS)

```
i18n.pluralization.cs = count => [
    count === 1
        ? 'one' : [2, 3, 4].indexOf(count) !== -1
            ? 'few' : 'other',
]
```

Questions?

Projects

🍺?

Sources

- https://github.com/wix/Detox/
- https://github.com/fnando/i18n-js