Speak React Native

React Native course by U+_

Sli.do (#K097)

https://app2.sli.do/event/obrkqsrj/questions

Your homework?

Overview

- IDE setup
- Display data sets
- Debugging
- Props
- State
- React Navigation

Project structure

Project structure

```
├── assets                             # Image assets
│   ├── icon.png
│   └── splash.png
├── bin
├── src                                # App source code (95 % of our work will be here)
│   ├── components                     # Components
│   │   ├── __tests__                  # Directory with test files for components
│   │   ├── RoundedButton.js
│   │   └── index.js
│   ├── containers                     # “App screens”
│   │   └── RootContainer.js
│   ├── themes                     # General styles and constants definition
│   │   ├── Colors.js                  # Defines colors in the app
│   │   ├── Metrics.js                 # Defines numeric values for dimensions in the app
│   │   └── index.js
│   └── types                          # Data types
│       └── index.js
├── App.js                             # Entry point for the app
├── app.json                           # Expo config
├── package.json                       # Project configuration – dependencies, scripts,...
└── yarn.lock                          # Generated file, ensures same library versions
```

IDE setup

IDE setup

- Settings for VS Code
    - https://github.com/jvaclavik/speak-react-native-skeleton/blob/master/.vscode/settings.json
- Plugins
    - Prettier – Code formatter
    - ESLint
    - EditorConfig for Visual Studio Code

Prettier

- Automatically formats code on save
- Enforces consistent codestyle
- Configuration file `.prettierrc`

```json
{
 "parser": "babylon",
 "printWidth": 80,
 "semi": false,
 "singleQuote": true,
 "trailingComma": "all",
 "bracketSpacing": true,
 "jsxBracketSameLine": false
}
```

ESLint

- Code standard rules
- Configuration file `.eslintrc`

```
{
   "rules": {
     "no-confusing-arrow": 0,
     "no-mixed-operators": 0,
     "no-nested-ternary": 0,
     "no-param-reassign": 0,
     "no-shadow": 0,
    }
}
```

EditorConfig

- Maintains consistent codestyles between different IDEs
- Configuration file `.editorconfig`
- https://editorconfig.org

```
indent_size = 2
indent_style = space
insert_final_newline = true
trim_trailing_whitespace = true
```

Debugging

Console log

```
const beers = ["Svijany", "Bernard", "Hendrych"]
console.log(beers)
console.log("beers: ", beers)
console.log(`beers: ${beers}`)
```

Run remote debugging

- Chrome debugging
- React Native Debugger

Chrome debugging

- Shake the device
    - iOS simulator `CMD + ALT + Z`
    - Android `CTRL(CMD) + M`
- “Debug remote JS”

React Native Debugger

- Standalone app
- Inspecting styles, network requests, ...
- Change port if you use Expo
  - Debugger -> New window -> enter port number

- https://github.com/jhen0409/react-native-debugger

Display data sets

Display data sets

- Map function
- FlatList

Our array

```
const beers = ["Svijany", "Bernard", "Hendrych"]
```

Map function in general

- Iterates over an array, applying given function to each item

```
const beers = ["Svijany", "Bernard", "Hendrych"]
beers.map(beerName => {
  return `${beerName} tastes good`
})
```

Map function in render

- Returns component for each item in dataset

```
const beers = ["Svijany", "Bernard", "Hendrych"]
...
render() {
  return (
    <View>
      {beers.map(beerName => (
        <Text>{beerName}</Text>
      ))}
    </View>
  )}
```

FlatList

- Generates an optimized listing component from array

```
const beers = ["Svijany", "Bernard", "Hendrych"]
...
render() {
  return (
    <FlatList
      data={beers}
      renderItem={{ item } => (
        <Text>{item}</Text>
      )}
    />
  )}
```

Keys

- Both `.map()` and `FlatList` should be using keys for performance reasons
- A key should be unique among siblings (not globally)
- Whenever possible, do NOT use array index as key, use ID or some other unique field instead

Keys

```
const beers = [
 {
   id: 1,
   name: "Svijany",
 },
 {
   id: 2,
   name: "Bernard",
 },
]
```

```
<View>
  {beers.map(beer => (
    <Text key={beer.id}>{beer.name}</Text>
  ))}
</View>

<FlatList
  data={beers}
  renderItem={({ item }) => <Text>{item.name}</Text>}
  keyExtractor={(item) => item.id}
/>
```

Keys

```
const beers = [
 {
   id: 1,
   name: "Svijany",
 },
 {
   id: 2,
   name: "Bernard",
 },
]
```

```
<View>
  {beers.map(beer => (
    <Text key={beer.id}>{beer.name}</Text>
  ))}
</View>

<FlatList
  data={beers}
  renderItem={({ item }) => <Text>{item.name}</Text>}
  keyExtractor={(item) => item.id}
/>
```

Props

Props

- Render function is called when props has changed
- Parameters (inputs) of the component
- Accessible through `this.props` or function parameters

Props

```
// Pass props to ChildComponent
export default class ParentComponent
extends React.PureComponent {
  render() {
    return
      <ChildComponent
        title="Title"
        count={123}
      />
  }
}
```

```
// Access parameters in ChildComponent
export default class ChildComponent
extends React.PureComponent {
  render() {
    return (
      <View>
        <Text>{this.props.title}</Text>
        <Text>{this.props.count}</Text>
      </View>
    )
  }
}
```

State

State

- To keep some data in component/container
- Only use `setState` for state updates, do not use `=`

```
this.setState({ someKey: updatedValue })
this.setState((prevState) => ({ someKey: updatedValue }))
```

Work with state (bad practice)

```
export default class Counter extends React.PureComponent {
 state = {
   counter: 0,
 }
 addOne = () => {
   this.setState({ counter: this.state.counter + 1 })
 }
 render() {
   return <Button onPress={this.addOne}>+1</Button>
 }
}
```

Work with state (bad practice)

```
export default class Counter extends React.PureComponent {
 state = {
   counter: 0,
 }
 addOne = () => {
   this.setState({ counter: this.state.counter + 1 })
 }
 render() {
   return <Button onPress={this.addOne}>+1</Button>
 }
}
```

Work with state (good practice)

```
export default class Counter extends React.PureComponent {
 state = {
   counter: 0,
 }
 addOne = () => {
   this.setState(prevState => ({ counter: prevState.counter + 1 }))
 }
 render() {
   return <Button onPress={this.addOne}>+1</Button>
 }
}
```

Work with state (good practice)

```
export default class Counter extends React.PureComponent {
 state = {
   counter: 0,
 }
 addOne = () => {
   this.setState(prevState => ({ counter: prevState.counter + 1 }))
 }
 render() {
   return <Button onPress={this.addOne}>+1</Button>
 }
}
```

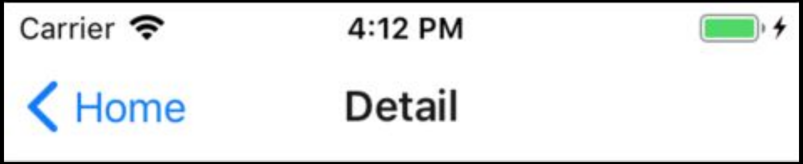
React Navigation

React Navigation

```
yarn add react-navigation
```

React Navigation

- Screen history (like in web browser)
- Transitions between screens
- Includes navbar, title, navbar buttons, back buttons



App.js (app entry point)

```
import React from "react"
import Navigator from "./src/containers/Navigator"

export default class App extends React.PureComponent {
 render() {
   return <Navigator />
 }
}
```

Navigator.js

```
import { createStackNavigator, createAppContainer } from
"react-navigation"
import RootContainer from "./RootContainer"
import Detail from "./Detail"

export default createAppContainer(createStackNavigator({
  Root: { screen: RootContainer },
  Detail: { screen: Detail },
}))
```

RootContainer.js

```
export default class RootContainer extends React.PureComponent {
  static navigationOptions = { title: "Home" }
  navigate = () => {
    this.props.navigation.navigate("Detail")
  }
  render() {
    ...
  }
}
```

RootContainer.js

```
render() {
  return (
    <SafeAreaView style={styles.container}>
      <RoundedButton onPress={this.navigate}>
        Navigate
      </RoundedButton>
    </SafeAreaView>
  )
}
```

Detail.js

```
export default class Detail extends React.PureComponent<null> {
  render() {
    return (
      <SafeAreaView style={styles.container}>
        <Text>Hello</Text>
      </SafeAreaView>
    )
  }
}
```

Pass a prop

```
this.props.navigation.navigate("Detail", {
  item: {
    image: "https://www.placecage.com/c/200/300",
  },
})
```

Access a navigation prop

```
export default class Detail extends React.PureComponent<null> {
  render() {
    const { item } = this.props.navigation.state.params
    return (
      <SafeAreaView style={styles.container}>
        <Image
          source={{ uri: item && item.image }}
        />
      </SafeAreaView>
    )
  }
}
```

Access a navigation prop

```
export default class Detail extends React.PureComponent<null> {
  render() {
    const { item } = this.props.navigation.state.params
    return (
      <SafeAreaView style={styles.container}>
        <Image
          source={{ uri: item && item.image }}
        />
      </SafeAreaView>
    )
  }
}
```

Homework

- Use tabs with stack navigator for each tab
  - https://reactnavigation.org/docs/en/tab-based-navigation.html
- First tab should contain list of items (use FlatList)
  - Click on item should open item detail
- Second tab should contain contact information
  - Picture, phone, address, ...

Questions?

🍺?

Sources

- https://reactjs.org/docs/lists-and-keys.html
- https://facebook.github.io/react-native/docs/flatlist
- https://reactnavigation.org/docs/en/params.html