Speak React Native

6

React Native course by U+_

Overview

- RxJS, Redux observable
- Epics + using with API
- Platform specific changes
- Styles

RxJS

- Reactive programming library
- Observable
  - core concept of the library
  - emits a stream of values, unlike promise, which resolves only once
  - something like function which can have more than one `return`
  - we subscribe to it, instead of calling it
  - can behave both synchronously and asynchronously

Creating observables

- We can use constructor or creation functions
- Most of the time, we will be using `from` function, which creates observable from nearly anything (Promise, Array, Iterable,...)

Adding to project

```
$ yarn add rxjs
$ yarn add redux-observable
```

Creating observables

```
import { from } from "rxjs"
import { getMovies } from "../api"

// this creates an observable from axios promise
const observable = from(getMovies())

// this is what it wouald look like using constructor
const observable2 = new Observable(observer => {
  getMovies().then(data => observer.next(data))
})
```

Creating observables

```
import { from } from "rxjs"
import { getMovies } from "../api"

// this creates an observable from axios promise
const observable = from(getMovies())

// this is what it would look like using constructor
const observable2 = new Observable(observer => {
  getMovies().then(data => observer.next(data))
})
```

RxJS operators

- Functions allowing us to work with data stream from an Observable
- For example map functions, which execute a callback on each value from the stream

RxJS operators

- `filter` - similar to Array.filter, lets us pick just some values from the stream
- `flatMap, switchMap` - both apply given function to each value in the stream
- `catchError` - similar to `.catch()`, serves for error handling

FlatMap vs. SwitchMap

- May behave differently when used in async code
- Both accept operations from the stream
- `FlatMap` - emits values as soon as operations are resolved (i.e. unknown order)
- `SwitchMap` - passes values in the order the operations arrived

FlatMap vs. SwitchMap - example

- User clicks a button multiple times and triggers multiple requests which will take different amount of time
- `FlatMap` - as soon as any of the requests is finished, emits value, even if the last request finished first
- `SwitchMap` - waits for the request which is on the turn and emits values in the order the requests arrived

FlatMap vs. SwitchMap - not clear?

- In case the difference is not clear, this article has a nice explanation: https://medium.com/@johnvoon/understanding-rxjs-and-redux-observable-93d953d436c6

RxJS operators

```
.pipe(
   filter(item => item.type === "type we want to pick"),
   flatMap(item => doSomething(item)),
   catchError(e => someErrorHandlingFunction(e))
)
```

RxJS operators

```
.pipe(
   filter(item => item.type === "type we want to pick"),
   flatMap(item => doSomething(item)),
   catchError(e => someErrorHandlingFunction(e))
)
```

Epics

- Function which accepts stream of redux actions and returns other redux actions using Observables
- We will use epics for handling asynchronous code like api calls

Redux observable

- Middleware for using RxJS with redux
- Allows us to subscribe to redux actions as if we were working with observables

Implementation

https://github.com/jvaclavik/speak-react-native-skeleton/tree/add-epics

App.js

```
import { combineEpics, createEpicMiddleware } from "redux-observable"

const epicMiddleware = createEpicMiddleware({
 dependencies: {},
})
middleware.push(epicMiddleware)
const store = createStore(...)
epicMiddleware.run(combineEpics(moviesRequestEpic))
```

App.js (more epics)

```
import { combineEpics, createEpicMiddleware } from "redux-observable"

const epicMiddleware = createEpicMiddleware({
 dependencies: {},
})
middleware.push(epicMiddleware)
const store = createStore(...)
epicMiddleware.run(combineEpics(moviesRequestEpic,movieDetailRequestEpic))
```

Update Movies redux

```
export const initialState = {
 loading: false,
 items: [],
 error: null,
}
```

Update Movies redux (actions)

```
export const onMoviesRequest = () => ({ type: "ON_MOVIES_REQUEST" })
export const onMoviesSuccess = movies => ({
  type: "ON_MOVIES_SUCCESS",
  movies,
})
export const onMoviesFail = error => ({
  type: "ON_MOVIES_FAIL",
  error,
})
```

Update Movies redux (reducers)

```
case "ON_MOVIES_REQUEST":
  return {
    ...state,
    loading: true,
    error: null,
  }
case "ON_MOVIES_SUCCESS":
  return {
    ...state,
    loading: false,
    items: action.movies.results,
  }
```

Update Movies redux (reducers)

```
case "ON_MOVIES_ERROR":
  return {
    ...state,
    loading: false,
    error: action.error,
  }
```

Movies epic

```
export const moviesRequestEpic = action$ =>
 action$.pipe(
   filter(action => action.type === "ON_MOVIES_REQUEST"),
   switchMap(() =>
     from(getMovies()).pipe(
       flatMap(response => from([onMoviesSuccess(response.data)])),
     ),
   ),
 )
```

Movie detail epic

```
export const movieDetailEpic = action$ =>
 action$.pipe(
   filter(action => action.type === "ON_MOVIE_DETAIL_REQUEST"),
   switchMap(action =>
     from(getMovie(action.movieId)).pipe( // getMovie is from api.js
       flatMap(response => from(onMovieDetailSuccess(response.data))),
       catchError(e => of(onMovieDetailFail(e))),
     ),
   ),
 )
```

Movie detail epic

```
export const movieDetailEpic = action$ =>
 action$.pipe(
   filter(action => action.type === "ON_MOVIE_DETAIL_REQUEST"),
   switchMap(action =>
     from(getMovie(action.movieId)).pipe( // getMovie is from api.js
       flatMap(response => from(onMovieDetailSuccess(response.data))),
       catchError(e => of(onMovieDetailFail(e))),
     ),
   ),
 )
```

Platform specific changes

Platform specific component

- Allows us to write platform-specific code

```
import { Platform } from "react-native"

<View>
  {Platform.OS === "ios" ?
    <IOSComponent />
    :
    <AndroidComponent />
  }
</View>
```

Platform specific styles

```
import { Platform } from "react-native"

const styles = StyleSheet.create({
 container: {
   backgroundColor: Platform.OS === "ios" ? "blue" : "red",
 },
})
```

Styles

Styles

- Defines design on the app
- Flat structure and styles separated for components
- JSON like & camelCase notation
- React Native Styles are not the same as CSS, but similar
- Less attributes than in CSS
  - E.g.: no "`float`" or "`text-align`"

Styles

- You can use JS code inside of styles

```
fieldErrorMessage: {
  ...errorText,
  fontSize: 18,
}
```

Styles (JS) vs. CSS

```
fieldErrorMessage: {
  fontSize: 18,
}
```

```
.fieldErrorMessage {
  font-size: '16px';
}
.fieldErrorMessage > a.primary {
  font-size: '16px';
}
```

StyleSheets

- You can work with styles as a plain object, but you shouldn’t do that
- Every render re-creates styles
- Use `StyleSheets.create` to pass references

StyleSheets

```
import { StyleSheet } from 'react-native'

// Good!
const styles = StyleSheet.create({
  button: {
    backgroundColor: "red",
  }
})
```

StyleSheets

```
import { StyleSheet } from 'react-native'

// Good!
const styles = StyleSheet.create({
  button: {
    backgroundColor: "red",
  }
})
```

```
// Bad!
const styles = {
  button: {
    backgroundColor: "red",
  }
}
```

StyleSheets with parameter

```
import { StyleSheet } from 'react-native'


const styles = (background) =>  StyleSheet.create({
  button: {
    backgroundColor: background,
  }
})
```

Flexbox

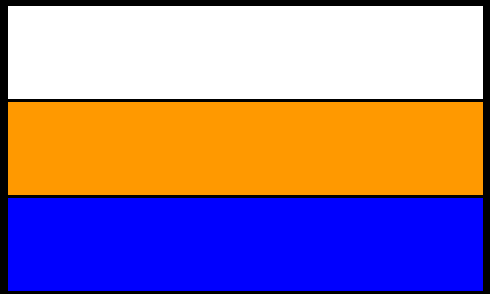
- CSS layout type
- Similar use as in CSS, but different implementation
- Defines how elements are visually related

Flexbox

- `flex: 1 ~~~~ width: "100%"`
- https://css-tricks.com/snippets/css/a-guide-to-flexbox/

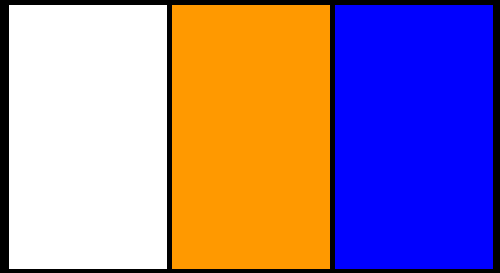
Flexbox (1:1:1)

```
<View style={{ flexDirection: 'column', flex: 1 }}>
  <View style={{ flex: 1, backgroundColor: 'white' }}></View>
  <View style={{ flex: 1, backgroundColor: 'orange' }}></View>
  <View style={{ flex: 1, backgroundColor: 'blue' }}></View>
</View>
```



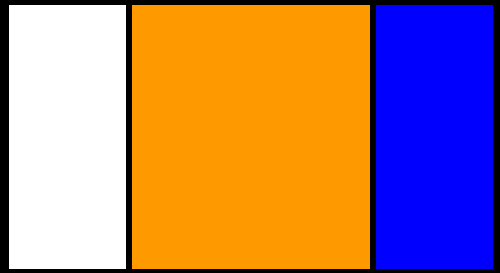
Flexbox (1:1:1)

```
<View style={{ flexDirection: 'row', flex: 1 }}>
  <View style={{ flex: 1, backgroundColor: 'white' }}></View>
  <View style={{ flex: 1, backgroundColor: 'orange' }}></View>
  <View style={{ flex: 1, backgroundColor: 'blue' }}></View>
</View>
```



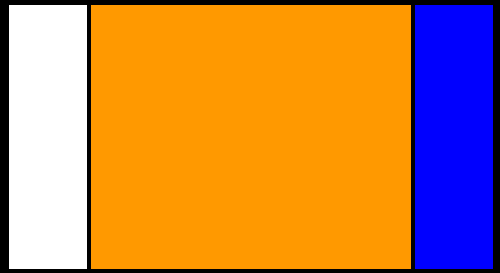
Flexbox (1:2:1)

```
<View style={{ flexDirection: 'row', flex: 1 }}>
  <View style={{ flex: 1, backgroundColor: 'white' }}></View>
  <View style={{ flex: 2, backgroundColor: 'orange' }}></View>
  <View style={{ flex: 1, backgroundColor: 'blue' }}></View>
</View>
```



Flexbox (1:4:1)

```
<View style={{ flexDirection: 'row', flex: 1 }}>
  <View style={{ flex: 1, backgroundColor: 'white' }}></View>
  <View style={{ flex: 4, backgroundColor: 'orange' }}></View>
  <View style={{ flex: 1, backgroundColor: 'blue' }}></View>
</View>
```



Styling tips

- Some properties don’t work well on both platforms, for example `overflow: 'visible'` on Android is buggy
- Working with image size can be tricky, sometimes you need to set `width: undefined` explicitly to make it work
- Use React Native debugger for styling

Questions?

Sources

- https://medium.com/@johnvoon/understanding-rxjs-and-redux-observable-93d953d436c6
- https://github.com/ReactiveX/rxjs/blob/master/doc/observable.md