



# React Native

Day 5



# Redux Saga

# Da nossa aula sobre Redux, sobrou uma pergunta...

- E quando uma mudança de estado produz um **efeito colateral**?

# O que é um efeito colateral?

Em geral, são coisas que acontecem **fora do escopo** da árvore de componentes do React Native

- **Buscar ou enviar dados** através de uma API
  - Buscar nome e avatar do GitHub
  - Carregar vídeos e imagens de um banco de dados
  - Enviar dados para cadastrar um usuário no backend
  - Verificar que um usuário realmente existe no ato de login

# Vimos que desencadeamos um efeito colateral...

- Com uma mudança de estado
- A ação ligada a essa mudança pode ou não pedir parâmetro

```
{type: GET_ACTION_MOVIES}
```

```
{type: GET_MOVIE_DETAILS, movie: 'Avengers: Endgame'}
```

```
{type: SIGNUP, email: user@gmail.com, password: '12345'}
```

Mas se o Redux só cuida de inserir e servir dados do estado central...

Quem cuida de buscar e receber dados de uma API, por exemplo?

# 0 Redux Saga

- Existem outras bibliotecas que também cuidam de efeitos colaterais e que são até mais simples de configurar e usar
  - Redux Thunk
- O Saga, no entanto, é a biblioteca que **escala melhor**
  - Quando seu app cresce **n**, a complexidade de gerenciar o Saga não cresce **n<sup>3</sup>**
- Mais **robusta** e com comportamento mais **estável**
- E como o ignite **já configura o Saga** para nós, não temos muita razão para não usá-lo

**Ação é disparada**




**Redux recebe a ação**




**Saga recebe a ação**





Mas para entender como o Saga funciona,  
precisamos primeiro entender um dos seus  
principais recursos



# ES6 Generators

# Quando você roda um programa JS, você assume:

- Que o programa executa em **um único thread**
- Seus comandos são executados **sequencialmente**
  
- Portanto: para que uma função execute, outra tem que terminar

```
// Declarar o loop
```

```
meuLoop = () => {  
  for (let i = 0; i < 100000000000; i++) {  
    console.log(i)  
  }  
}
```

```
setTimeout(() => console.log("Hello, world!"), 5) // 5 ms
```

```
meuLoop() // Chamar o loop
```

Resultado:

0

...

100000000000

Hello, world!

# Generators

- E se a gente pudesse **interromper o fluxo** do loop para imprimir “Hello, World” e depois voltar para o loop?
- Generators são funções que possuem o poder de **interromper o próprio fluxo** quando necessário
  - Nesse caso, seria o **loop** que decide ceder controle da execução do programa momentaneamente para o **setInterval**

# Regras

- Generators possuem **sintaxe especial** para diferenciá-la de funções normais
- Eles interrompem o **próprio** fluxo, e nunca são interrompidos por **outras funções**
- Algum **gatilho** tem que ser disparado para devolver a execução do programa ao generator
- Quando a execução é **interrompida**, dados são **enviados**
- Quando a execução **resume**, dados são **retornados**

```
function *foo() {  
  yield 1  
  yield 2  
  yield 3  
}
```

```
let it = foo()
```

```
console.log( it.next() ) // {value: 1, done: false}  
console.log( it.next() ) // {value: 2, done: false}  
console.log( it.next() ) // {value: 3, done: false}  
console.log( it.next() ) // {value: undefined, done: true}
```



```
function *foo(x) {  
  let y = yield (x + 1)  
  let z = 2 + yield (y / 3)  
  yield z  
}
```

```
let it = foo(5)
```

```
console.log( it.next() )    // {value: 6, done: false}  
console.log( it.next(18) ) // {value: 6, done: false}  
console.log( it.next(2) )  // {value: 4, done: false}  
console.log( it.next() )   // {value: undefined, done: true}
```

# O que isso tem a ver com Saga?

- Chamadas a APIs tendem a ser demoradas
- Caso seu APP parasse de funcionar toda vez que você fizesse uma chamada a uma API, ele travaria com frequência
- Um saga é uma função generator que executa as chamadas à API e enquanto a API não envia o retorno, ela pausa sua própria execução e entrega o controle para o resto do programa
- Quando a resposta chega, ela resume sua execução

```
// Generators
// Saga
// Apisause
// Redux

// {type: GET_USER_AVATAR, username: 'aryella18'}

export function * getUserAvatar (api, action) {

  const { username } = action
  const response = yield call(api.getUser, username)

  if (response.ok) {
    const { avatar } = response.data
    yield put(GithubActions.putUserAvatar(avatar))
  } else {
    yield put(GithubActions.error("Avatar not found"))
  }
}
```

# Notaram algo de interessante?

- Cada Saga interage tanto com o Redux em dois momentos:
  - GET / VERIFY / SIGNUP
    - **Uma ação lança uma comunicação com a API**
    - Dados relevantes para a chamada à API são buscadas do estado central
  - PUT / RESULT
    - **A resposta da API lança uma ação**
    - Dados relevantes da resposta da API são inseridas no estado central



Apisauce



# Como definir sua comunicação com a API?

- Felizmente, isso é a parte mais fácil
- Usamos o Apisauce, que **centraliza suas chamadas a APIs**
- Você precisa saber somente o ponto de acesso (endpoint) da API, as rotas que você quer usar, e como essas rotas devem ser chamadas
- Isso você geralmente encontra na documentação da API

## Root endpoint

You can issue a `GET` request to the root endpoint to get all the endpoint categories that the REST API v3 supports:

```
curl https://api.github.com
```

## Get a single user ⓘ

Provides publicly available information about someone with a GitHub account.

The `email` key in the following response is the publicly visible email address from your GitHub [profile page](#). When setting up your profile, you can select a primary email address to be “public” which provides an email entry for this endpoint. If you do not set a public email address for `email`, then it will have a value of `null`. You only see publicly visible email addresses when authenticated with GitHub. For more information, see [Authentication](#).

The Emails API enables you to list all of your email addresses, and toggle a primary email to be visible publicly. For more information, see "[Emails API](#)".

```
GET /users/:username
```



## Response

Status: 200 OK

```
{
  "login": "octocat",
  "id": 1,
  "node_id": "MDQ6VXNlcjE=",
  "avatar_url": "https://github.com/images/error/octocat_happy.gif",
  "gravatar_id": "",
  "url": "https://api.github.com/users/octocat",
  "html_url": "https://github.com/octocat",
  "followers_url": "https://api.github.com/users/octocat/followers",
  "following_url": "https://api.github.com/users/octocat/following{/other_user}",
  "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/octocat/starred{/owner}/{repo}",
  "subscriptions_url": "https://api.github.com/users/octocat/subscriptions",
  "organizations_url": "https://api.github.com/users/octocat/orgs",
  "repos_url": "https://api.github.com/users/octocat/repos",
  "events_url": "https://api.github.com/users/octocat/events{/privacy}",
  "received_events_url": "https://api.github.com/users/octocat/received_events",
  "type": "User",
  "site_admin": false,
  "name": "monalisa octocat",
  "company": "GitHub",
  "blog": "https://github.com/blog",
  "location": "San Francisco",
  "email": "octocat@github.com",
  "hireable": false,
  "bio": "There once was...",
  "public_repos": 2,
  "public_gists": 1,
  "followers": 20,
  "following": 0,
  "created_at": "2008-01-14T04:33:35Z",
  "updated_at": "2008-01-14T04:33:35Z"
}
```

```
import apisauce from 'apisauce'
```

```
const create = (baseUrl = 'https://api.github.com/') =>  
{
```

```
  // Etapa 1
```

```
  const api = apisauce.create({  
    baseUrl,  
    headers: { 'Cache-Control': 'no-cache' },  
    timeout: 10000  
  })
```

```
  // Etapa 2
```

```
  const getUser = (username) => {  
    return api.get(`/users/${username}`)  
  }
```

```
  // Etapa 3
```


```
  return {  
    getUser  
  }
```

```
}
```


```
export default {
```

```
  create
```

```
}
```



Agora vamos implementar uma tela de  
cadastro com acesso a uma API de  
verificação de email



# Referências

[Redux Saga](#)

[Apisauce](#)

[Generators](#)