Professor Rodrigo Bossini
https://www.rodrigobossini.com.br

ReactJS
Redux

1 Exercícios

Considere uma instituição de ensino que possui o seguinte funcionamento.

Departamento de Vestibular

1. Pessoas interessadas em se matricular realizam um vestibular. Para tal, elas informam seu nome e cpf.

2. Quando uma pessoa faz o vestibular, ela tem uma nota final atribuída, que varia de 0 a 10. Cada pessoa tem 70% de chance de tirar uma nota entre 6 e 10.

3. A instituição tem um departamento para armazenamento do histórico de vestibular. Cada entrada no histórico tem nome, cpf e nota final do candidato a que se refere.

Departamento de matrícula

1. O departamento de matrícula permite que alunos aprovados no vestibular se matriculem. Ao se matricular, um aluno informa apenas o seu cpf.

2. Alunos somente podem ser matriculados caso tenham sido aprovados no vestibular.

3. O departamento de matrícula armazena um histórico de todos os alunos que tentam fazer matrícula. Cada entrada no histórico possui o cpf de um aluno e o seu status. Alunos que tentam se matricular e que não foram aprovados no vestibular, têm status "NM", de não matriculado. Alunos que tentam se matricular e que foram aprovados no vestibular, têm o status "M", de matriculado.

Produção: outubro de 2021

Escreva uma função de teste que oferece as seguintes opções.

1. Realizar vestibular. Esta funcionalidade captura nome e cpf de um candidato e, a seguir, simula a realização da prova do vestibular, armazenando seus dados e um valor gerado – neste momento – aleatoriamente variando no intervalo real [0, 10]. Lembre-se de seguir a distribuição de probabilidade estipulada.

2. Realizar matrícula. Esta funcionalidade captura o cpf de um candidato e, a seguir, tenta fazer a sua matrícula. A sua matrícula somente ocorre caso ele tenha sido considerado aprovado no vestibular. Aprovados são aqueles que têm nota maior ou igual a seis. Lembre-se de controlar os status "M" e "NM".

3. Visualizar meu status. Esta função captura o cpf de um candidato e exibe seu status na lista de matrículas.

4. Visualizar lista de aprovados. Esta função exibe os dados de todos os alunos aprovados no vestibular.

0. Sair do sistema.

Faça a implementação utilizando a biblioteca Redux.

Respostas

1 (Novo projeto) Crie uma pasta para abrigar a sua solução. Com um terminal vinculado a ela, use

npm init -y

para criar o arquivo package.json. Use

code .

para abrir uma instância do VS Code vinculada à pasta. No VS Code, clique Terminal >> New Terminal para obter um terminal interno.

2 (Dependências) Instale as seguintes dependências. Usaremos o pacote **prompts** para obter dados digitados pelo usuário.

npm install redux
npm install prompts
npm install --save-dev nodemon

3 (Novo arquivo e script de execução) Crie um arquivo chamado **index.js**. A seguir, abra o arquivo **package.json** e adicione o script destacado no Bloco de Código 3.1. Ele viabilizará a execução do projeto com **npm start**.

Bloco de Código 3.1

```
{
  "name": "sistema_vestibular_matricula",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "prompts": "^2.4.2",
    "redux": "^4.1.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.14"
  }
}
```

No terminal interno do VS Code, use

npm start

para colocar o aplicativo em execução. Se desejar, clique Terminal >> New Terminal novamente para obter outro terminal interno do VS Code.

4 (Criador de ação: prestar vestibular) O primeiro criador de ação que definiremos cria uma ação que representar uma prova de vestibular realizada. Veja o Bloco de Código 4.1.

Bloco de Código 4.1

```
const redux = require ('redux')
const prompts = require ('prompts')

// função criadora de ação
// nome e cpf serão capturados via prompt
const realizarVestibular = (nome, cpf) => {
  // nota gerada aleatoriamente
  const entre6e10 = Math.random() <= 0.7;
  const nota = entre6e10 ? Math.random() * 4 + 6 : Math.random() * 5;
  // esse JSON é uma ação
  return {
    type: "REALIZAR_VESTIBULAR",
    payload: {
      nome,
      cpf,
      nota
    }
  };
};
```

5 (Criador de ação: realizar matrícula) O criador de ação do Bloco de Código 5.1 cria uma ação que representa uma tentativa de matrícula.

Bloco de Código 5.1

```
// função criadora de ação
// cpf capturado via prompt
// status de acordo com a realização do vestibular
const realizarMatricula = (cpf, status) => {
  // esse JSON é uma ação
  return {
    type: "REALIZAR_MATRICULA",
    payload: {
      cpf, status
    }
  };
};
```

6 (Reducer: "Fatia" do estado que representa o histórico de vestibular) O reducer do Bloco de Código 6.1 se encarrega de receber uma ação que representa um vestibular realizado e armazená-la no histórico.

Bloco de Código 6.1

```
// essa função é um reducer
const historicoVestibular = (historicoVestibularAtual = [], acao) => {
  if (acao.type === "REALIZAR_VESTIBULAR") {
    return [...historicoVestibularAtual, acao.payload];
  }
  return historicoVestibularAtual;
};
```

7 (Reducer: "Fatia" do estado que representa o histórico de matrículas) O Bloco de Código 7.1 mostra um reducer responsável por manipular o histórico de matrículas.

Bloco de Código 7.1

```
// essa função é um reducer
const historicoMatriculas = (historicoMatriculasAtual = [], acao) => {
  if (acao.type === "REALIZAR_MATRICULA") {
    return [...historicoMatriculasAtual, acao.payload];
  }
  return historicoMatriculasAtual;
};
```

8 (Combinando os reducers e criando o store do Redux) Lembre-se que o objeto "store" do Redux é uma abstração que engloba diferentes funções – incluindo os reducers - e o estado. Para construir um store, primeiro combinamos os reducers com **combineReducers**. Entregamos o seu resultado para a função **createStore**, produzindo assim o objeto store. Veja o Bloco de Código 8.1.

Bloco de Código 8.1

```
const todosOsReducers = redux.combineReducers({
  historicoMatriculas,
  historicoVestibular
});
const store = redux.createStore(todosOsReducers);
```

9 (Função de teste) A função de teste oferecerá um menu para o usuário. Ele poderá escolher entre realizar vestibular, realizar matrícula, visualizar seu status de matrícula e visualizar a lista de aprovados no Vestibular. Veja a sua definição inicial no Bloco de Código 9.1.

Bloco de Código 9.1

```
// função de teste
const main = async () => {
  const menu =
    "1-Realizar Vestibular\n2-Realizar Matrícula\n3-Visualizar meu status\n4-Visualizar lista de aprova-
dos\n0-Sair"
  let response;
  do {
    response = await prompts({
        type: 'number',
        name: 'op',
        message: menu
    });
    try {
      switch (response.op) {
        case 1:{
            break;
        }
        case 2:{
            break;
        }
        case 3:{
             break;
        case 4:{
             break;
        case 0:
          console.log("Até logo");
          break;
        default:
          console.log("Opção inválida");
          break;
      }
    } catch (err) {
      console.log(err)
      console.log("Digite uma opção válida");
    }
  } while (response.op !== 0);};
```

Na opção 1, capturamos nome e cpf do usuário e, a seguir, fazemos dispatch de uma ação para realização do vestibular. Veja o Bloco de Código 9.2.

Bloco de Código 9.2

```
...
case 1:{
            const {nome} = await prompts({
                type: 'text',
                name: 'nome',
                message: "Digite seu nome"
            });
            const {cpf} = await prompts({
                type: 'text',
                name: 'cpf',
                message: "Digite seu cpf"
            });
            store.dispatch(realizarVestibular(nome, cpf));
            break;
        }
...
```

Na opção 2, capturamos o cpf do usuário e verificamos se ele está aprovado. Se estiver, fazemos a sua matrícula e ele fica com status "M" no histórico. Caso contrário, ele fica com status "NM" no histórico. Veja o Bloco de Código 9.3.

Bloco de Código 9.3

```
...
case 2:{
          const {cpf} = await prompts({
              type: 'text',
              name: 'cpf',
              message: "Digite seu cpf"
          });
          const aprovado = store.getState().historicoVestibular.find(
                    aluno => aluno.cpf === cpf && aluno.nota >= 6)
          if (aprovado){
              store.dispatch(realizarMatricula(cpf, "M"))
              console.log ("Ok, matriculado!")
          }
          else{
              store.dispatch(realizarMatricula(cpf, "NM"))
              console.log ("Infelizmente você não foi aprovado no
                                                  vestibular ainda.")
          }
            break;
        }
...
```

Na opção 3, capturamos apenas o cpf do usuário. Se ele estiver matriculado ou se já tiver tentado se matricular, mostrado o seu status, que pode ser M ou NM. Caso contrário, mostramos uma mensagem que indica que ele não existe no histórico de matrículas. Veja o Bloco de Código 9.4.

Bloco de Código 9.4

```
...
case 3:{
          const {cpf} = await prompts({
              type: 'text',
              name: 'cpf',
              message: "Digite seu cpf"
          });
          const aluno = store.getState().historicoMatriculas.find(
                                              aluno => aluno.cpf === cpf)
          if (aluno){
              console.log (`Seu status é: ${aluno.status}`)
          }
          else{
              console.log("Seu nome não consta na lista de matrículas")
          }
          break;
       }
...
```

Na opção 4 mostramos a lista de aprovados no vestibular. Ou seja, aqueles que tiveram nota maior ou igual a 6. Veja o Bloco de Código 9.5.

Bloco de Código 9.5

```
...
case 4:{
           const listaAprovados =
           store.getState().historicoVestibular.filter(aluno => aluno.nota >= 6);
           console.log(listaAprovados);
           break;
        }
...
```

Ao final, apenas chamamos a função main. Veja o Bloco de Código 9.6.

Bloco de Código 9.6

```
...
const main = async () => {
  ...
};
```

main()

Referências

React – A JavaScript library for building user interfaces. 2021. Disponível em <https://reactjs.org/>. Acesso em agosto de 2021.

Redux - A predictable state container for JavaScript apps. | Redux. 2021. Disponível em <https://redux.js.org>. Acesso em outubro de 2021.