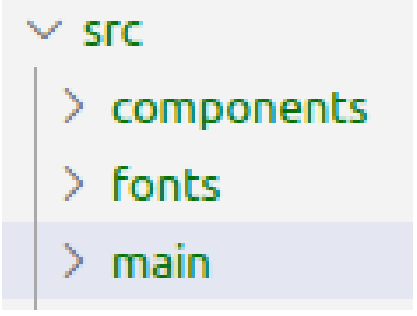
Desenvolvimento de Software para WEB

Aula 02
Projeto Robô
Projeto Calculadora

Projeto Robô

Preparação do Projeto

- Crie um novo projeto:
  - create-react-app robo
- Dentro do projeto, organize as pastas:



Preparação do Projeto

- Baixe a seguinte fonte (RobotoMono):
  - https://fonts.google.com/specimen/Roboto+Mono?selection.family=Roboto+Mono
  - Descompacte o arquivo “RobotoMono-Thin.ttf” dentro da past “fonts” do projeto.

Parte 1
Criando a Classe Principal

Classe Robô

- Dentro da pasta main, crie os seguintes arquivos:
  - Robo.jsx
  - Robo.css

Classe Robô

- Robo.jsx

```
import React,{Component} from 'react'
import './Robo.css'

export default class Robo extends Component{

    render(){
        return(
            <div className='robo'>

            </div>
        )
    }
}
```

O arquivo que contem a classe CSS a ser usada por este componente!

Essa é a classe CCS que iremos criar no próximo slide, como o nome “.robo”. Aqui, não usamos o ponto “.”

Classe Robô

- Robo.css
  - Praticamente, toda classe que criarmos, terá associada a ela um CSS. O padrão é manter os nomes todos iguais.

```
.robo {
    height: 320px;
    width: 235px;
    border-radius: 5px; /* arredondamento na bordas */
    overflow: hidden; /*https://www.w3schools.com/css/css_overflow.asp*/
    background-color: azure;
}
```

Classe Robô

```
@font-face {
  font-family: 'RobotoMono'; /* nome qualquer */
  src: url('./fonts/RobotoMono-Thin.ttf'); /* onde está a fonte */
} /* https://developer.mozilla.org/pt-BR/docs/Web/CSS/@font-face */

* {
  font-family: 'RobotoMono', monospace;
} /* define que todas as classes usarão essa fonte */

body {
  display: flex; /* https://www.w3schools.com/css/css3_flexbox.asp */
  height: 100vh; /* vh-> 1/100 da altura do navegador */
  justify-content: center;
  align-items: center;
  text-align: center;

  color: #fff;
  background-color: #c0c0c0;
}
```

- index.css

Também devemos alterar o CSS de index, modificando assim o estilo de TODAS as páginas da aplicação.

Classe Robô

- index.js

```
import Robo from './main/Robo'
import * as serviceWorker from './serviceWorker';

ReactDOM.render(
    <div>
        <h2>Robô Simples</h2>
        <Robo />
    </div>

    ,document.getElementById('root')
);
```

Em index.js, apenas iremos chamar a classe Robo, considerada principal.

Parte 2
Os Botões de Controle

Os Botões de Controle

- Os botões servirão para manipular o robô no plano (x,y)
- Na pasta components, crie o arquivos:
  - Button.jsx
  - Button.css

Os Botões de Controle

- Button.jsx

```
import React from 'react'
import './Button.css'

export default props=>{

    return(
        <button className={button}>
            {props.label}
        </button>
    )
}
```

Mesmo esquema da classe Robo.

Os Botões de Controle

- Button.css

```css
/* criação de variáveis */
:root {
    --bg-button: #f0f0f0;
    --border-button: solid 1px #888;
}

.button {
    font-size: 1.4em;
    background-color: var(--bg-button);
    border: var(--border-button);
    outline: none;
}
```

Não é necessária a criação de variáveis. No entanto, é interessante que elas existam, pois podem ser usadas por outras classes CSS.

Os Botões de Controle

- Robo.jsx

```
import React,{Component} from 'react'
import './Robo.css'
import Button from '../components/Button'

export default class Robo extends Component{

    render(){
        return(
            <div className='robo'>
                <Button label='UP' />
                <Button label='LEFT' />
                <Button label='RIGHT' />
                <Button label='DOWN' />
            </div>
        )
    }
}
```

Os botões devem servir para modificar a posição do robô no plano (x,y).

Os Botões de Controle

- Robo.css

```
.robo {
    height: 320px;
    width: 235px;
    border-radius: 5px; /* arredondamento na bordas */
    overflow: hidden; /*https://www.w3schools.com/css/css_overflow.asp*/
    background-color: azure;

    display: grid;
    grid-template-columns: repeat(2,50%);
    grid-template-rows: 1fr 50px 50px 50px;
}
```

A principal modificação é o template em grid.

Parte 3
Criando o Display

Display

- No Display, iremos dar um feedback visual da posição do robô na grid.
- Na pasta components, crie os arquivos:
  - Display.jsx
  - Display.css

Display

- Display.jsx

```
import React from 'react'
import './Display.css'

export default props=>{
    return(
        <div className='display'>{props.value}</div>
    )
}
```

Display

- Display.css

```
.display {
    grid-column: span 2; /* vai ocupar duas colunas*/
    background-color: #0004;
    color: black;
    font-weight: bold;

    display: flex;
    justify-content: center;
    align-items: center;
    padding: 10px;
    font-size: 0.5em;
    overflow: hidden;
}
```

Parte 4
Lógica do Botões

Lógica

- Button.css
  - subclasses de .button

```
.button:active {
    background-color: #ccc;
}

/*remove outline pontilhado no firefox*/
.button::-moz-focus-inner {
    border: 0;
}

.button.double {
    grid-column: span 2;
}
```

Lógica

- Button.jsx

```
import React from 'react'
import './Button.css'

export default props=>{

    let classes = 'button '
    classes += props.double ? 'double' : ''

    return(
        <button className={classes}
                onClick={()=>props.click && props.click(props.label)}>
            {props.label}
        </button>
    )
}
```

Se o props.double for verdade, adiciona-se o CSS à "classes".

Lógica

- Robo.jsx

```
setPosition(position){
    let i = this.state.i
    let j = this.state.j
    if(position === 'UP'){
        i = i - 1
    }else if(position === 'DOWN'){
        i = i + 1
    }else if(position === 'LEFT'){
        j = j - 1
    }else if(position === 'RIGHT'){
        j = j + 1
    }
    this.setState({i,j}) }
```

Lógica

- Robo.jsx

```
constructor(props){
    super(props)
    this.state = {i:5,j:10}

    this.setPosition = this.setPosition.bind(this)

}
```

Lógica

- Robo.jsx

```
drawMatrix(){
    let out = ''
    for (let i = 0; i < 14; i++) {
        for(let j = 0; j < 14; j++){
            if(i === this.state.i && j === this.state.j)
                out+='[O]'
            else
                out+='[X]'
        }

    }
    return out
}
```

Lógica

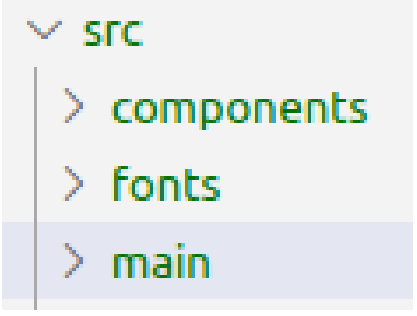
- Robo.jsx

```
render(){
    return(
        <div className='robo'>
            <Display value={this.drawMatrix()}/>
            <Button label='UP' click={this.setPosition} double/>
            <Button label='LEFT' click={this.setPosition} />
            <Button label='RIGHT' click={this.setPosition}/>
            <Button label='DOWN' click={this.setPosition} double/>
        </div>
    )
}
```

Projeto Calculadora

Preparação do Projeto

- Crie um novo projeto:
  - create-react-app calculator
- Dentro do projeto, organize as pastas:



Preparação do Projeto

- Baixe a seguinte fonte (RobotoMono):
  - https://fonts.google.com/specimen/Roboto+Mono?selection.family=Roboto+Mono
  - Descompacte o arquivo “RobotoMono-Thin.ttf” dentro da past “fonts” do projeto.

Parte 1
Criando a Classe Principal

Calculator

- Dentro de main, crie o arquivo Calculator.jsx e Calculator.css.
- Como Calculator será um componente com estado, iremos usar classes.
- Calculator.css será o arquivo de folha de estilo para Calculator.

Calculator

- Calculator.jsx

```
import React,{Component} from 'react'
import './Calculator.css'

export default class Calculator extends Component{
    render(){
        return(
            <div className='calculator'>
                teste
            </div>
        )
    }
}
```

Calculator

- Calculator.css

```
.calculator {
    height: 320px;
    width: 235px;
    border-radius: 5px; /* arredondamento na bordas */
    overflow: hidden; /*https://www.w3schools.com/css/css_overflow.asp*/
    background-color: red;
}
```

Alterando o index.*

- index.css

```css
@font-face {
  font-family: 'RobotoMono'; /* nome qualquer */
  src: url('./fonts/RobotoMono-Thin.ttf'); /* onde está a fonte */
} /* https://developer.mozilla.org/pt-BR/docs/Web/CSS/@font-face */

* {
  font-family: 'RobotoMono', monospace;
} /* define que todas as classes usarão essa fonte */

body {
  display: flex; /* https://www.w3schools.com/css/css3_flexbox.asp */
  height: 100vh; /* vh-> 1/100 da altura do navegador */
  justify-content: center;
  align-items: center;
  text-align: center;

  color: #fff;
  background-color: #c0c0c0;
}
```

Alterando o index.*

- index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import Calculator from './main/Calculator';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(
    <div>
        <h2>Calculator</h2>
        <Calculator />
    </div>
...
```

Parte 2
Criando os Botões

Button

- Crie a pasta “components”.
- Dentro de “components”, crie os arquivos:
  - Button.css
  - Button.jsx

Button

- Buton.jsx

```
import React from 'react'
import './Button.css'

export default props=>
    <button className='button'>{props.label}</button>
```

Button

- Button.css

```
:root {
    --bg-button: #f0f0f0;
    --border-button: solid 1px #888;
}

.button {
    font-size: 1.4em;
    background-color: var(--bg-button);
    border: none;
    border-right:  var(--border-button);
    border-bottom: var(--border-button);
    outline: none;
}
```

Button

- Calculator.jsx

```
...
 render(){
        return(
            <div className='calculator'>
                <Button label='AC'/>
                <Button label='/'/>
                <Button label='7'/>
                <Button label='8'/>
                <Button label='9'/>
                <Button label='*'/>
                <Button label='4'/>
                <Button label='5'/>
                <Button label='6'/>
                <Button label='-'/>
                <Button label='1'/>
                <Button label='2'/>
                <Button label='3'/>
                <Button label='+'/>
                <Button label='0'/>
                <Button label='.'/>
                <Button label='='/>
            </div>
...
```

Button

- Calculator.css

```
.calculator {
    height: 320px;
    width: 235px;
    border-radius: 5px; /* arredondamento na bordas */
    overflow: hidden; /*https://www.w3schools.com/css/css_overflow.asp*/
    background-color: red;

    /*para o final*/
    display: grid;
    grid-template-columns: repeat(4,25%);
    grid-template-rows: 48px 48px 48px 48px 48px /*repeat(5,48px)*/
}
```

Parte 3
Componente Display

Display

- O Display irá mostrar a entra e a saída da calculadora.
- Crie, em components, os seguintes arquivos:
  - Display.jsx
  - Display.css

Display

- Display.jsx

```
import React from 'react'
import './Display.css'

export default props=>
    <div className='display'>{props.value}</div>
```

Display

- Display.css

```
.display {
    grid-column: span 4;
    background-color: #0004;

    display: flex;
    justify-content: flex-end;
    align-items: center;
    padding: 20px;
    font-size: 2.0em;
    overflow: hidden;
}
```

Display

- Calculator.css

```
...
grid-template-rows: 1fr 48px 48px 48px 48px 48px /*repeat(5,48px)*/
...
```

Parte 4
Melhorando os Botões

Melhorando os Botões

- Os próximos passos são:
  - criar estilos para diferentes tipos de botões (operações e dígitos)
  - capturar, via função interna, o botão clicado (seu label)

Melhorando os Botões

- Button.css

```
.button:active {
    background-color: #ccc;
}

/*remove outline pontilhado no firefox*/
.button::-moz-focus-inner {
    border: 0;
}

.button.double {
    grid-column: span 2;
}

.button.triple {
    grid-column: span 3;
}

.button.operation {
    background-color: #fa8231;
    color: #fff;
}

.button.operation:active {
    background-color: #fa8231cc;
```

Melhorando os Botões

- Button.jsx

```
export default props=>{

    /* escolhe a classe css de acordo com o que foi passado pelo props */
    let classes = 'button '
    classes += props.operation ? 'operation':''
    classes += props.double ? 'double':''
    classes += props.triple ? 'triple':''

    return (
        <button
            /* chama uma função click() vinda do pros */
            /* passa pra click() o valor do label do botão apertado */
            onClick={()=>props.click && props.click(props.label)}
            className={classes}>
            {props.label}
        </button>
    )
}
```

Melhorando os Botões

```
export default class Calculator extends Component{

    constructor(props){
        super(props)

        this.clearMemory = this.clearMemory.bind(this)
        this.setOperation = this.setOperation.bind(this)
        this.addDigit = this.addDigit.bind(this)

    }

    clearMemory(){
        console.log('limpar')
    }

    setOperation(operation){
        console.log('operação: '+operation)
    }

    addDigit(n){
        console.log('dígito: '+n)
    }
```

- Calculator.jsx

Melhorando os Botões

```
render(){
    return(
        <div className='calculator'>
            <Display value='100'/>
            <Button label='AC' click={this.clearMemory} triple={true}/>
            <Button label='/' click={this.setOperation} operation/>
            <Button label='7' click={this.addDigit}/>
            <Button label='8' click={this.addDigit}/>
            <Button label='9' click={this.addDigit}/>
            <Button label='*' click={this.setOperation} operation/>
            <Button label='4' click={this.addDigit}/>
            <Button label='5' click={this.addDigit}/>
            <Button label='6' click={this.addDigit}/>
            <Button label='-' click={this.setOperation} operation/>
            <Button label='1' click={this.addDigit}/>
            <Button label='2' click={this.addDigit}/>
            <Button label='3' click={this.addDigit}/>
            <Button label='+' click={this.setOperation} operation/>
            <Button label='0' click={this.addDigit} double/>
            <Button label='.' click={this.addDigit}/>
            <Button label='=' click={this.setOperation} operation/>
        </div>
    )
  }
}
```

- Calculator.jsx

Parte 5
A lógica da função addDigit

Lógica addDigit

- Inicialmente, em Calculadora.jsx, deve-se criar um objeto que armazene o estado inicial da aplicação.

```
...
const initialState = {
    displayValue: '0',
    clearDisplay: false,
    operation: null,
    values: [0,0],
    current:0
}

export default class Calculator extends Component{
...
```

Lógica addDigit

```
....
 constructor(props){
     super(props)

     this.clearMemory = this.clearMemory.bind(this)
     this.setOperation = this.setOperation.bind(this)
     this.addDigit = this.addDigit.bind(this)

     this.state = {...initialState}
 }

 clearMemory(){
     //console.log('limpar')
     this.setState({...initialState})
 }
```

O construtor, função clearMemory e o render devem refletir o estado inicial da Calculadora

```
...
return(
    <div className='calculator'>
        <Display value={this.state.displayValue}/>
```

Lógica addDigit

```
addDigit(n){
        //console.log('dígito: '+n)
        /* caso o display já tenha um número com ponto '.', essa regra
        impede que mais pontos sejam adicionados ao número */
        if(n==='.' && this.state.displayValue.includes('.'))
            return

        /* clearDisplay será true caso o display tenha APENAS 0 ou o valor do
          state.clearDisplay for true */
        let clearDisplay = false
        if(this.state.displayValue==='0' || this.state.clearDisplay)
            clearDisplay = true

        /* o currentValue  vai depender do clearDisplay */
        let currentValue = ''
        if(clearDisplay) currentValue = ''
        else currentValue = this.state.displayValue
```

Lógica addDigit

```
    ...
    /* concatenando o currentValue com o que foi digitado */
    let displayValue = currentValue + n

    /* atualizando o display. clearDisplay é falso para poder sempre
    concatenar com um novo digito */
    this.setState({displayValue,clearDisplay:false})

    /* só é válido entrar aqui se for um digito */
    if(n !== '.'){
        const i = this.state.current //qual digito no array será trabalhado
        const newValue = parseFloat(displayValue) //transformando o display
        const values = [...this.state.values] //copiando os valores
        values[i] = newValue //colocando o valor dentro do array
        this.setState({values}) //atualizando o array
        console.log(values)
    }
}
```

Parte 6
A função setOperation

Lógica do setOperation

```
setOperation(operation){

    //console.log('operação: '+operation)
    if(this.state.current === 0)
        // se eu ainda estiver no índice 0 do meu array de valores
        // devo apagar meu display e setar o índice para 1,
        // além de armazenar a operação
        this.setState({operation,current:1,clearDisplay:true})
    else{
        let equals = false
        // se a opeção selecionada for o igual, seto a variável
        // equal pra true (eu finalizei o cálculo e devo mostrar algo)
        if(operation === '=')
            equals = true
        const currentOperation = this.state.operation
```

Lógica do setOperation

```
// se forem uma das operações matemáticas abaixo, eu efetuo o
// cálculo e armazeno no índice 0 e depois zero o índice 1
const values = [...this.state.values]
if(currentOperation === '+'){
    values[0] = values[0]+values[1]
}else if(currentOperation === '-'){
    values[0] = values[0]-values[1]
}else if(currentOperation === '*'){
    values[0] = values[0]*values[1]
}else if(currentOperation === '/'){
    values[0] = values[0]/values[1]
}
values[1]=0
```

Lógica do setOperation

```
        this.setState({
            displayValue:values[0], // mostre o valor do índice 0
            operation: equals ? null : operation, // se equal for verdadeiro, não tem operação
            current: equals ? 0 : 1, // algum cálculo foi feito e deve-se voltar ao 0
            clearDisplay: !equals, // não limpa o display se o equals for true
            values // copia novos valores
        })

    }
}
```

Referências

- Exemplos baseados em:
  - https://github.com/cod3rcursos/curso-react-redux