



UNIVERSIDADE CATÓLICA DE ANGOLA  
FACULDADE DE ENGENHARIA

# ROSAURA FRAMEWORK DOCUMENTAÇÃO



## Sumário

INTRODUÇÃO .....	4
Como Funciona .....	4
Componentes BuildComponent .....	6
childElements: .....	6
events: .....	7
styles: .....	7
css: .....	7
ref: .....	7
type: .....	7
Exemplo de reutilização de componentes .....	7
Component Button.js .....	7
Arquivo App.js .....	8
Componentes nativos do Framework .....	9
DATABINDING .....	9
Databing Variáveis De Estado .....	9
defVariable .....	9
varPrint para imprimir variáveis de estado em atributos text .....	10
CONTROLADORES DE ESTADO .....	11
varMonitor .....	11
paramsMonitor .....	11
routeIn .....	11
routeLeave .....	12
ATRIBUTO REF DO BUILDCOMPONENT .....	13
ROTAS .....	14
REGRAS .....	15
Componentes .....	15
Assets .....	15
Execução .....	15
entre rotas. ....	15
COMANDO NPM PARA INSTALAR .....	15
REPOSITÓRIO GIT DO FRAMEWOR .....	16



## INTRODUÇÃO

O Rosaura é um framework JavaScript desenvolvido para facilitar o desenvolvimento de aplicações web dinâmicas e interativas. Ele oferece um conjunto de componentes e utilitários que permitem gerenciar o estado da aplicação, manipular o DOM de forma eficiente e criar interfaces responsivas.

### Como Funciona

A principal chamada do arquivo index, responsável por referenciar o elemento pai #root, via chamada do BuildRootComponent, que coloca os elementos filhos dentro da div#root

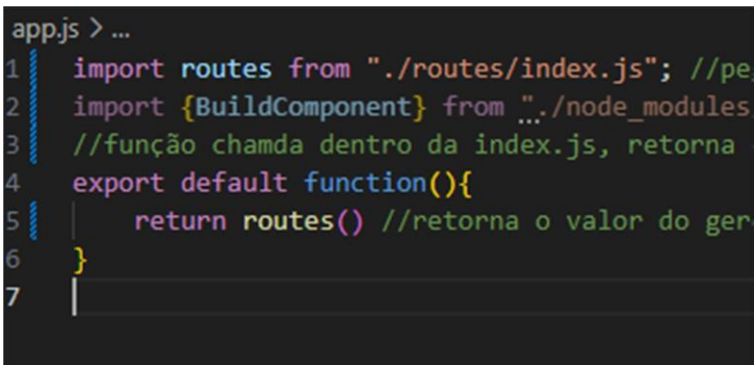
A screenshot of a code editor showing the content of index.js. The code imports Rosaura from a local node\_modules path and an app function from app.js. It then defines a main function that calls Rosaura.BuildRootComponent with app() and finally calls main().

```
import Rosaura from "../node_modules/rozaaura/index.js"; //R
import app from "../app.js"; //app é a função que retorna o

//funç
//usa const main: () => void } root elemento, e atribui o
const main =function(){
  Rosaura.BuildRootComponent(app()) //Instancia o root
}
main() //chamada damain
```

Figura 1 index.js

Em seguida temos a função app.js. que se encarrega por ser intermediário entre a rota e a index.js

A screenshot of a code editor showing the content of app.js. The code imports routes from routes/index.js and BuildComponent from node\_modules. It then exports a default function that returns routes().

```
app.js > ...
1 import routes from "../routes/index.js"; //pe
2 import {BuildComponent} from "../node_modules
3 //função chamda dentro da index.js, retorna
4 export default function(){
5   return routes() //retorna o valor do ger
6 }
7
```

Figura 2. app.js

As rotas são criadas em routes/index.js

Os componentes, devem estar obrigatoriamente em src, para cada componente deve existir uma pasta, se necessitar de inclusão de arquivo css, deve-se, na mesma pasta criar o index.css, e depois, no BuildComponent, fazer a chamada via função useCss, como argumento da função o caminho relativo das pestas, apartir da primeira pasta dentro de src, até a última pasta do arquivo css.

```

> .js index.js > default
import Rosaura from "../../node_modules/rozaaura/index.js"
import Page1 from "../../src/Pages/Page1/index.js"
import Page2 from "../../src/Pages/Page2/index.js"
💡
export default () => {

  const [title, setTitle] = Rosaura.defVariable("")
  return (
    //O gerenciador de rotas
    Rosaura.RosauraRouter (
      //Instanciador de rotas, em array, e indexador de ids
      Rosaura.RosauraRoute(
        {
          routes: [
            {element: Page1, path: "/page1", attributes: {}},
            {element: Page2, path: "/page2", attributes: {}}
          ]
        }
      )
    )
  )
}

```

Figura 3. routes/index.js

Cada rota solicita o objecto, contendo o elemento BuildComponent, o caminho e os atributos do elemento, como acima, tem-se o elemento Page1

```

import Rosaura, {BuildComponent, useFile, varPrint, useCss} from "../../node_modules/rozaaura/index.js"
import Card from "../../Components/Card/index.js"
import LeftBar from "../../Components/LeftBar/index.js"
import Title from "../../Components/Title/index.js"
import Message from "../../Message/index.js"

export default ({routerref, title, setTitle}) => {

  return(
    BuildComponent({
      style: {flexDirection: 'row'},
      childElements: [
        LeftBar(),
        BuildComponent({
          style: { padding: '30px', flex: 1},
          childElements: [
            Title({text: varPrint("Página 1 🍌")}),
            Message()
          ]
        })
      ]
    })
  )
}

```

Figura 4. Component Page1

## Componentes BuildComponent

Os componentes são elementos cruciais, eles são gerados por meio da chamada de BuildComponent.

BuildComponent é utilizado para criar elementos HTML dinamicamente com base nas configurações passadas.

Ex:

```
import routes from "./routes/index.js"; //pega o valor retornado
do gerenciador de rotas
import {BuildComponent} from "./node_modules/rozaaura/index.js"
//função chama dentro da index.js,
retorna component export default
function(){      return BuildComponent({
text: "Hello, world"
    })
}
```

### Parâmetros BuildComponentes

```
BuildComponent({
  childElements =
  [],      text,
  style = {},
  events = {},
  attributes =
  {},
  type="div",
  ref = {},
  css})
```

#### childElements:

- Dentro do BuildComponent, uma lista de elementos filho (childElements) é criada usando várias chamadas de BuildComponent. Cada elemento possui diferentes atributos, estilos, e conteúdo dinâmico utilizando varPrint para exibir o valor de v.

**events:**

- São os eventos, devem ser chamados sem a utilização da inicial on.

**styles:**

- São os estilos, devem ser passados como objectos, no caso de espaço entre palavras devem ser substituído pela notação camelCase.

**css:**

- recebe o valor de retorno do useCss.

**ref:**

- recebe como valor uma variável do tipo defVariable, e referencia dinamicamente o elemento criado

**type:**

- é opcional, define o tipo htmlElement

**Exemplo de reutilização de componentes**

Imagina-se criar dois botões com as mesmas características, só que um verde e outro azul.

**Component Button.js**

Para teste, tem-se o seguinte componente.

```
const Button = ({text, style})
=> {
  return(
    BuildComponent({
      type: "button",
      text,
      style,
    })
  )
}
```

## Arquivo App.js

Aqui está uma implementação do arquivo app.js, com um componente!

```
//função chamada dentro da index.js,  
retorna component  
  
export default function(){  
  return BuildComponent({  
    text:  
    "Hello, world",  
    childElements: [  
      Button({text: "Botão um", style:  
        {backgroundColor: "green", color: "white"}}),  
  
      Button({text: "Botão dois", style:  
        {backgroundColor: "blue", color: "white"}}),  
  
    ],  
  
    style: {  
      gap: '20px',  
      alignItems: 'center',  
      justifyContent:  
        'center',  
      height: '100%',  
      fontSize: '30px'  
    }  
  })  
}
```



## Componentes nativos do Framework

BuildComponent:

- É genérico

InpuComponent:

- Próprio para inputs

FormComponent:

- Próprio para formulários

## DATABINDING

O Framework Rosaura usa two-way databing, ele foi implementado de forma a dar muitas possibilidades

### Databing Variáveis De Estado

defVariable

defVariable é uma função que mapea a mudança de estado de uma variável.

Para acessar o valor da variável via código, acessa-se o atributo value da variável, como mostrado asseguir.

```
const [valor, setValor] = defVariable()

varMonitor( _ => {
  console.log(valor.value)
}, [valor])
```

Para imprimir o valor na página usa-se o varPrint, exemplo

Imprimir dados no innerHTML página dinamicamente

Considere, contar cliques em um botão e mostrar na página

```

/função chama dentro da index.js, retorna
component export default function(){

    const [b1, setB1] = defVariable(0)
    return BuildComponent({
        text: varPrint("Clicaste {}
vezes na página ", [b1]),
        childElements: [

        Button({
            text:varPrint("clicou {}",[b1]),
            events: {
                click:_=> setB1(b1.value + 1)}
            })),
        ],
    })
}

```

Esse binding é possível em todos os aspectos anível de funcionalidade, mesmo em atributos de estilo, como em style do BuildComponent

O defVariable, gera um objecto, mapeado por uma função

### **varPrint para imprimir variáveis de estado em atributos text**

Ela mapea e imprime uma lista de variáveis de estado em cada { }

```

import Rosaura, {BuildComponent, defVariable, varMonitor,
    useCss} from ".././../node_modules/rozaaura/index.js"

export default ({}) => {

    const [a, setA] = defVariable()
    const [b, setB] = defVariable()
    const [c, setC] = defVariable()

    varMonitor( () => {
        console.log("Se uma das tres mudar!")
    }, [a,b,c])

    return(
        BuildComponent({
            //varPrint
            text: varPrin("Clica para mudar a cor {} {} {} ", [a,b,c]),
        })
    )
}

```

## CONTROLADORES DE ESTADO

### varMonitor

É usado para monitorar variáveis de estado, e executar função sempre que a variável mudar.

### paramsMonitor

É usado para monitorar os parâmetros da url, chamado dentro de uma pagina de rota

```
import Rosaura, {BuildComponent, useFile, varPrint, useCss,
paramsMonitor} from "../../node_modules/rozaaura/index.js"

export default () => {

    paramsMonitor((object) => {
console.log(" Parametros mudaram ",
object)
    })
return(
    BuildComponent({
style:{flexDirection:
'row'},
childElements:
```

### routeIn

É chamada quando a rota é aberta, recebe dois parâmetros, uma é a função a ser chamada e o outro é o valor saído do parametro do arquivo de rota, o parâmetro é o routerref.

Dentro do Arquivo Home.js

```
import Rosaura, {BuildComponent, useFile, varPrint, routeIn,
paramsMonitor} from "../../node_modules/rozaaura/index.js"

export default ({routerref}) => {

    routeIn(() => {
console.log("Entrou na rotaa ")    },

    {routerref})
```

## routeLeave

É chamada quando a rota é fechada, similar ao `routeIn`, recebe o valor da variável `routerref`

```
import Rosaura, {BuildComponent, useFile, varPrint, routeLeave,
paramsMonitor} from
"../../../../../node_modules/rozaaura/in
dex.js" export default
({routerref}) => {

    routeLeave(() => {
console.log("Saiu da rota ")
    }, {routerref})
```

## ATRIBUTO REF DO BUILDCOMPONENT

Este atributo é importante para referenciar o elemento, usando variável de estado, ele seta o campo element, que contem o elemento referenciado dentro da variável de estado.

Exemplo de uma implementação, aproveitando o varMonitor

```
import Rosaura, {BuildComponent, defVariable, varMonitor,
  useCss} from "../../../../../node_modules/rozaaura/index.js"

export default ({}) => {

  const [valor, setValor] = defVariable()

  varMonitor( _ => {
    if(!valor?.element)
      return
    valor.element.style.color = valor.value
  }, [valor])

  return(
    BuildComponent({
      text: "Clica para mudar a cor",
      ref: valor,
      events: {
        click: parametro => {
          setValor("red")
        }
      }
    })
  )
}
```

O varMonitor, ajuda num binding mais eficiente, caso queira-se algo mais detalhado.

No exemplo acima, como o valor da variável será setado para luis, e nunca mudará a seguir, então a função dentro do varMonitor será chamada uma só vez.

OBS: o parametro da função click, faz referência ao objecto event, que contem o atributo target, de referência do elemento que sofre evento, assim como no html/js puro

## ROTAS

As rotas devem ser criadas em routes/index.js

```
import Rosaura from "../node_modules/rozaaura/index.js"
import Page1 from "../src/Pages/Page1/index.js"
import Page2 from "../src/Pages/Page2/index.js"

export default () => {

  const [title, setTitle] = Rosaura.defVariable("")
  return (
    //O gerenciador de rotas
    Rosaura.RosauraRouter (
      //Instanciador de rotas, em array, e indexador de ids
      Rosaura.RosauraRoute(
        {
          routes: [
            {element: Page1, path: "/page1", attributes: {}},
            {element: Page2, path: "/page2", attributes: {}}
          ]
        }
      )
    )
  )
}
```

RosauraRoute, devolve um vector de rotas mapeadas em um objecto, é útil, pois atribui um id, dinamicamente em cada rota

RosauraRouter, é o instanciador de rotas, ela gerencia as rotas, escolhendo qual abrir, os parâmetros trocados, a rota aberta e fechada, a chamada direia via browser, a navegação dos botões de recuo e avanço do navegador.

Links de secção, são gerenciados em RosauraRouter

## REGRAS

### Componentes

Os Componentes externo, devem estar isolados em uma pasta dentro de src, junto de um index.css se necessário

### Assets

Os arquivos externos, de multimédia, imagem ou texto, devem estar dentro da pasta assets. A função é useFile, exemplo

```
import Rosaura, {BuildComponent, useFile, varPrint, routeLeave, paramsMonitor} from
"../../node_modules/rozaaura/index.js" export
default ({routerref}) => {

  routeLeave() => {
console.log("Saiu da rota ")
  }, {routerref})
return(
  BuildComponent({      style:
{flexDirection: 'row'},
childElements: [
    LeftBar(),
    BuildComponent({
type: "img",
attributes: {
      src: useFile("nome_da_imagem_em_assets.jpg")
    }
  })
  ]
})
}
```

### Execução

É aconselhável o uso do routeIn e routLeave, quando for garantir independência total entre rotas.

Uso do defVariable é excencial, melhor do que variáveis comuns.

Rotas

Os links de rotas, devem começar sempre com /.

### COMANDO NPM PARA INSTALAR

Npm install rozaaura

## **REPOSITÓRIO GIT DO FRAMEWOR**

<https://github.com/luis687687/rosaura.git>