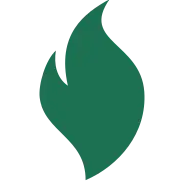


**UNIVERSIDADE CATÓLICA DE ANGOLA**

**FACULDADE DE ENGENHARIA**

**ROSAURA FRAMEWORK**

DOCUMENTAÇÃO



# 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 chamade do **BuildRootComponent,** que coloca os elementos filhos dentro da div#root

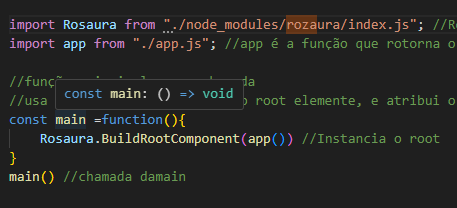


Figura 1 index.js

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

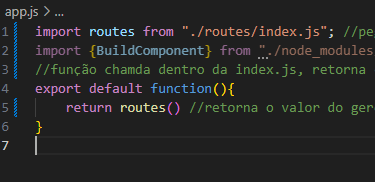


Figura 2. app.js

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.



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

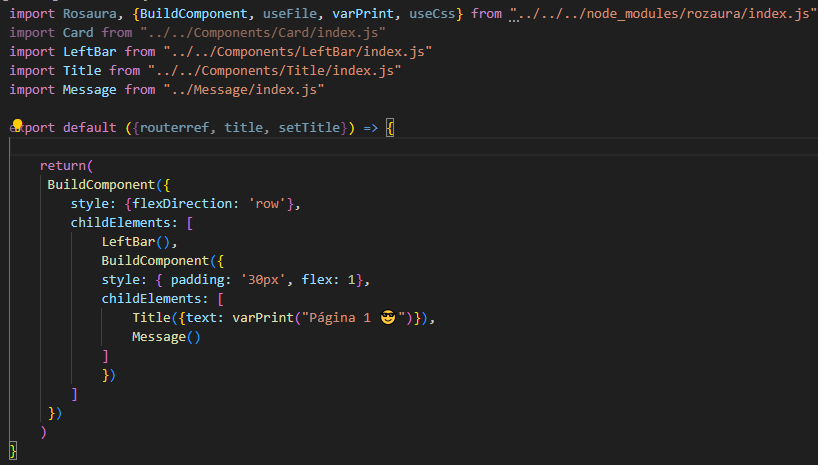


Figura 4. Component Page1

# 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 gereneciador de rotas

import {BuildComponent} from "./node\_modules/rozaura/index.js"

//função chamda 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 substituido 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 implementão do arquivo app.js, com um componente!

//função chamda dentro da index.js, retorna component

export default function(){

    return BuildComponent({

        text: "Hello, world",

        childElements: [

            Button({text: "Butão um", style:

{backgroundColor: "green", color: "white"}}),

            Button({text: "Butão 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 attributo value da variável, como mostrado asseguir.

const [v, setV] = defVariable()

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 chamda 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 anivel de funcionalidade, mesmo em atributos de estilo, como em style do BuildComponent

# CONTROLADORES DE ESTADO

## varMonitor

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

## paramsMonitor

É usado para monitorarar os parámetros da url

## routeIn

É chamada quando a rota é aberta

## routeLeave

É chamada quando a rota é fechada

# ROTAS

As rotas devem ser criadas em routes/index.js

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

RosauraRouter, é o instanciador de rotas, ela gerencia as rotas, escolhendo qual abrir, os parámetros 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 gerênciados 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.

## Execução

É aconcelhável 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 rozaura

# REPOSITÓRIO GIT DO FRAMEWOR

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