**MANUAL TÉCNICO**

**Elizabeth Carreño Alvarez**

**Carlos Hernán Gómez Gómez**

**Universidad Nacional De Colombia**

**Administración**

**Administración De Sistemas Informáticos**

**Manizales**

**2021**

[**RESUMEN**](#_tmye4sx39oc3) **3**

[**INTRODUCCIÓN**](#_286cyhxgpbfc) **4**

[**PASOS PARA EL DESARROLLO DE LA CH-MÁQUINA**](#_frml1ki9y3nj) **5**

[**ELABORACIÓN DEL CH-MÁQUINA**](#_ss965epwu2b) **5**

## 

## 

## RESUMEN

El CH-MÁQUINA posee cualidades que permiten la simulación del funcionamiento de un sistema operativo ficticio, como lo son la ejecución de programas (sólo programas con extensión .ch), tiene un espacio para la memoria que es manejado para programas, se presentan las diferentes instrucciones que están siendo ejecutadas para cada programa, es posible conocer los valores de los registros de los programas que se encuentran cargados en la memoria, el valor que va adquiriendo el acumulador y la instrucción que está en ejecución.

Cuenta con opciones de visualización de los resultados entregados durante la ejecución del programa que simulan una pantalla y una impresora.

Para desarrollar este proyecto utilizaremos Javascript es un **lenguaje de programación**, o lo que es lo mismo, un mecanismo con el que podemos decirle a nuestro navegador que tareas debe realizar. Javascript nos ofrece una mayor flexibilidad y un abanico de posibilidades más grande, y que bien usadas, pueden ahorrarnos bastante tiempo.

Vue js es un **framework progresivo** Javascript para crear interfaces de usuario, o en otras palabras, se trata de una «capa» añadida a Javascript formada por herramientas, convenciones de trabajo y un lenguaje particular que nos permite crear aplicaciones de forma rápida, agradable, sencilla y muy práctica.

Con el fin de poder desarrollar la aplicación CH-MÁQUINA, se hace necesario contar con herramientas como:

1. Nodejs es un entorno de ejecución para JavaScript

## PASOS PARA EL DESARROLLO DE LA CH-MÁQUINA

Para la creación del proyecto tendremos en cuenta herramientas que facilitaran la creación de la misma.

**Vite** se define como una herramienta de frontend que ayuda a crear tus proyectos de forma agnóstica y que su desarrollo y construcción final sea lo más sencilla posible

Para comenzar, simplemente necesitamos tener instalado **NodeJS**. Se utiliza npm para comenzar el proyecto.

Este comando ya se encargará de crear la carpeta para el proyecto, se debe indicar un parámetro opcional --template para especificar directamente la plantilla a utilizar, se escribe en el terminal:

npm init @vitejs/app ch-web --template vue

Con esto ya se tiene el proyecto listo para arrancarlo con el script

npm run dev

y comenzar a desarrollar. Ten en cuenta que en todos los proyectos que generes tendrás un script dev que lanza el **modo de desarrollo de Vite** y, por otro lado, tendrás un script build que lanza el **modo de producción de Vite.**

**En index.html** es la estructura html, se realiza llamado a **main.js**

<template>

<div class="ss-page">

<pre>{{ mainMemoryVariables }}</pre>

loads info

<pre>{{ loadsInfo }}</pre>

Instructions

<pre>{{ instructions }}</pre>

<label>

<input v-model="processType" type="radio" value="FCFS" />

FCFS

</label>

<label>

<input v-model="processType" type="radio" value="SJF" />

SJF

</label>

<label>

<input v-model="processType" type="radio" value="SPN" />

SPN

</label>

<label>

<input v-model="processType" type="radio" value="SRTN" />

SRTN

</label>

<label>

<input v-model="processType" type="radio" value="NO\_EXPROPIATIVE" />

Prioridad no expropiativo

</label>

<label>

<input v-model="processType" type="radio" value="EXPROPIATIVE" />

Prioridad expropiativo

</label>

<p>

const priorities = reactive([]);

const processType = ref("FCFS");

//necesita ser asincrona, debido a que la reactividad impide ver los cambios al tener paso a paso activo

async function onExecute() {

//para detener la reactividad por 1 milisegundo

const delay = (ms) => new Promise((res) => setTimeout(res, ms));

let instructionsLength = 0

// loadsInfo permite saber saber el identificador del proceso

//fileIndex fichero de cada proceso

for (let fileIndex = 0; fileIndex < loadsInfo.length; fileIndex++) {

const loadInfo = loadsInfo[fileIndex];

const startIndex = instructionsLength;

instructionsLength += loadInfo.linesLength;

//saber las instrucciones a que proceso pertenecen

const instructionsSection = instructions.value?.slice(

startIndex,

instructionsLength,

);

for (let index = 0; index < instructionsSection.length; index++) {

const instructionString = instructionsSection[index];

if (isStepByStep.value) {

await delay(1);

alert(`Desea continuar? ${instructionString}`);

}

const parts = instructionString.split(" ");

const instruction = parts[0];

// funcion a ejecutar

const action = functions[instruction];

if (action) {

// debugger

const newIndex = action(parts, index, fileIndex); //ejecutar la funcion

if (newIndex !== null && newIndex !== undefined) {

index = newIndex;

}

}

}

}

}

// la reactividad permite cambiar el valor de la constante y sus dependencias, accion en cadena, su referencia a la userMemory (proxi) no va a cambiar

Link a github: <https://github.com/ElizabethCA94/ch-web/commit/b9532566f45285e87100e7e5b128841570f9f92f>