Universidad San Carlos de Guatemala
Organización de Lenguajes y Computadores 1

Manual técnico

Nataly Saraí Guzmán Duarte

Carnet: 202001570

Índice

Introducción 3
Información destacada 3
Objetivos y alcances del sistema 4
Requerimientos 4
Datos técnicos para la realización del sistema 4
Lógica del programa 5
Utilización de codigo: 12
Diagrama general de la aplicación: 13

Introducción

El presente documento se adjuntan las características, funcionamiento y estructuras que fueron requeridas para el desarrollo del sistema presentado el cual consta de un sistema interprete de lenguaje SQL que da una tabla de símbolos y reportes y árbol ast, crea tablas en este lenguaje, etc.

Lugar de realización: Guatemala

Fecha: 28/10/2023

Responsable de elaboración: Nataly Saraí Guzmán Duarte

Información destacada

El programa fue realizado utilizando como plataforma visual studio code, la realización de la fase fue de aproximadamente 4 semanas, cuya realización fue separada en partes del menú, cada parte se realizo en aproximadamente 5 días su realización completa.

Objetivos y alcances del sistema

1. Aprender a crear analizadores léxicos y sintácticos utilizando JISON.
2. Manejar errores léxicos y sintácticos durante la interpretación.
3. Implementar el patrón de diseño intérprete en JavaScript.
4. Reconocer y procesar instrucciones en lenguaje SQL, incluyendo DDL y DML.
5. Gestionar diversos tipos de datos en las expresiones del lenguaje.
6. Garantizar la correcta precedencia de las operaciones en las expresiones.

Requerimientos

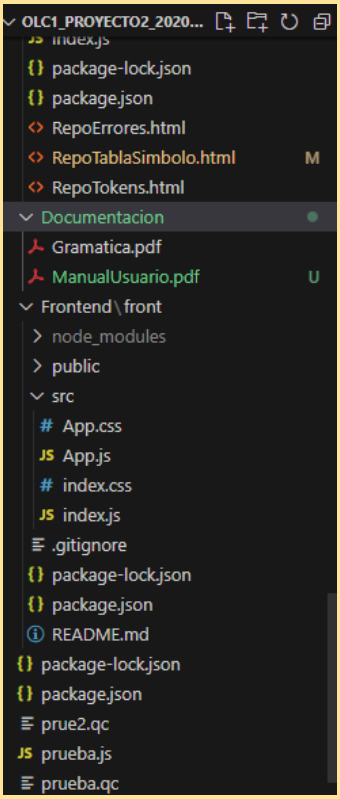
- RAM: 1 GB (mínimo).
- ROM: 250 MB (mínimo).
- Arquitectura x32 bits o x64 bits.
- Sistema operativo: Windows, Linux, MacOS.
- Lenguaje de programación: Javascript.
- Plataforma IDE: visual studio code.

Datos técnicos para la realización del sistema

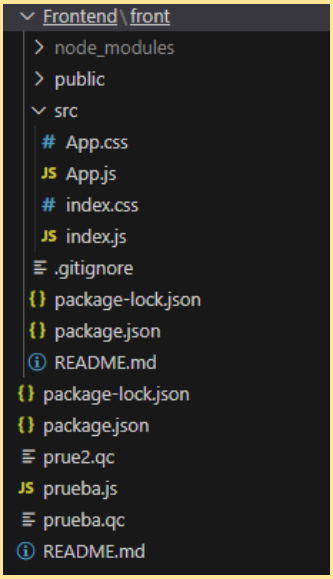
- Windows 10 Home Single Language.
- Procesador: Intel Core i5.
- RAM: 8.00 GB
- Plataforma IDE: visual studio code.

Lógica del programa

El programa está constituido por intefaz grafica creada utilizando react y una api creada en js, es simple e intuitivo con el usuario mostraremos a continuación las distintas clases que se usaron para su creación.



El proyecto se basa en el Frontend y en Backend; el frontend fue realizado con react y una plantilla que fue personalizada por la estudiante en html.



En el archivo app.js se basa todo el codigo del frontend y la conexión con la base de datos. Se utilizo axios para la conexión.

Para la utilización de axios primero debemos importar Axios al principio de tu archivo de componente o donde planeas realizar la solicitud. Luego ya se pueden realizar la solicitud. Para realizar una solicitud HTTP, utiliza una de las funciones proporcionadas por Axios, como get(), post(), put(), o delete(). Dentro del .then(), puedes acceder a la respuesta exitosa a través del objeto response. En el .catch(), puedes manejar errores que puedan ocurrir durante la solicitud. Esto es importante para manejar problemas como la falta de conexión a Internet, respuestas de error del servidor, etc.

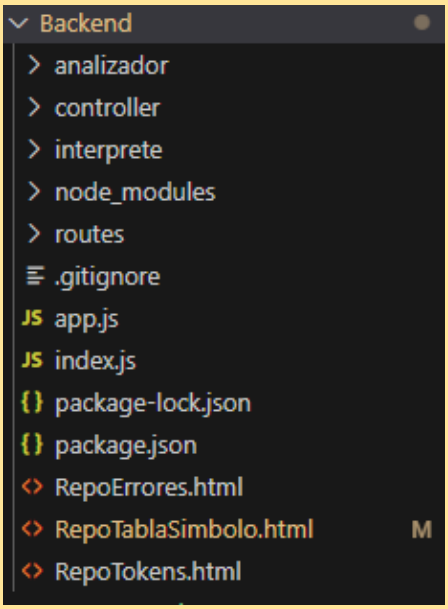
De esta manera realizamos la conexión:

```
import axios from 'axios';
import React, { useState, useEffect } from 'react';

// Tu código de componente aquí

var nombre;

function App() {
  const [data,setdata]= useState([]);
  useEffect(() => {
  }, [])

  function RepoTokens(event){
    event.preventDefault();
    axios.get("http://localhost:4000/repoto")
    .then((response)=>{
      setdata(response.data.data)
    })
  }      You, 5 days ago • Case simple
  function RepoErrores(event){
    event.preventDefault();
    axios.get("http://localhost:4000/repoerror")
    .then((response)=>{
      setdata(response.data.data)
    })
  }
  function RepoTaSim(event){
    event.preventDefault();
    axios.get("http://localhost:4000/reposim")
    .then((response)=>{
      setdata(response.data.data)
    })
```

Tambien utilizamos funciones para crear, abrir, guardar y eliminar archivos y que se puedan mostrar en el área de entrada, asi tambien al realizar el análisis que se pueda ver el resultado del interprete en el área de salida.

```
function Eliminar(event){
  event.preventDefault();
  document.getElementById('entrada').value = '';
}
function Guarda(event){
  event.preventDefault();
  var documento = document.getElementById('entrada').value;
  const blob = new Blob([documento], { type: 'text/plain' });
  const url = window.URL.createObjectURL(blob);
  const a = document.createElement('a');
  a.href = url;
  a.download = nombre;
  a.textContent = nombre;
  //document.body.appendChild(a);
  a.click();
  window.URL.revokeObjectURL(url);
}
function CrearArch(event){
  event.preventDefault();
  const contenido = ' ';
  var nombres = document.getElementById('nombrear').value;
  const blob = new Blob([contenido], { type: 'text/plain' });
  const url = window.URL.createObjectURL(blob);
  const a = document.createElement('a');
  a.href = url;
  a.download = nombres;
  a.textContent = nombres;
  //document.body.appendChild(a);
  a.click();
  window.URL.revokeObjectURL(url);
```

Ahora en el lado del backend, fue realizado con js creando una api simple.

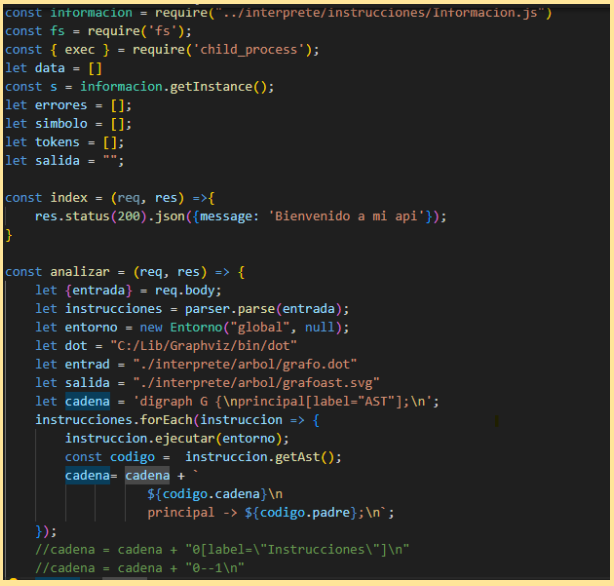


Contamos con diferentes carpetas y todo comienza con el index, para la creación de la api y los llamados a los get y post que se utilizaron para mandar la información al fronted.

La carpeta analizador contiene el archivo gramática.json donde se crearon el analizador léxico y sintactico, con la utilización de palabras reservadas, expresiones regulares, expresión sin retorno como los comentarios; y el analizador sintactico con la descripción de la gramática que se especifica en el archivo de gramáticas. Al correr y verificar que se hayan cumplido las reglas tanto en el

analizador sintactico como en el analizador léxico se genera el parser.js que es el que ya se puede utilizar con las demás clases.

Luego continuamos con la carpeta controller en esta se encuentra un archivo que realiza todas las conexiones, utilizando los get y set, tanto para analizar los archivos como para crear reportes.

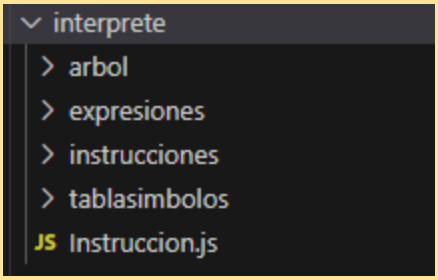


```
const informacion = require(`../interprete/instrucciones/Informacion.js`)
const fs = require('fs');
const { exec } = require('child_process');
let data = []
const s = informacion.getInstance();
let errores = [];
let simbolo = [];
let tokens = [];
let salida = "";

const index = (req, res) =>{
    res.status(200).json({message: 'Bienvenido a mi api'});
}

const analizar = (req, res) => {
    let {entrada} = req.body;
    let instrucciones = parser.parse(entrada);
    let entorno = new Entorno("global", null);
    let dot = "C:/Lib/Graphviz/bin/dot"
    let entrad = "./interprete/arbol/grafo.dot"
    let salida = "./interprete/arbol/grafoast.svg"
    let cadena = 'digraph G {\nprincipal[label="AST"];\n';
    instrucciones.forEach(instruccion => {
        instruccion.ejecutar(entorno);
        const codigo =  instruccion.getAst();
        cadena= cadena + `
                    ${codigo.cadena}\n
                    principal -> ${codigo.padre};\n`;
    });
    //cadena = cadena + "0[label=\"Instrucciones\"]\n"
    //cadena = cadena + "0--1\n"
```

Luego nos encontramos con la carpeta interprete, que tiene subcarpetas como lo son las expresiones, los instrucciones, tablasimboles y el archivo instrucción.js , en este archivo se encuentra la base de los demás archivos, se muestra las funciones como ejecutar, y llamar el geast.

En la carpeta de expresiones nos encontramos con una variedad de archivos que el archivo grmatica.jison hace un llamado para guardar los datos y asi poder obtenerlos para luego utilizarlo en las instrucciones; hay diferentes tipos de datos como ids, datos, aritmética, lógicos, casteos y entre otros, estos forman una estructura y son retornados para poder utilizarlos luego.

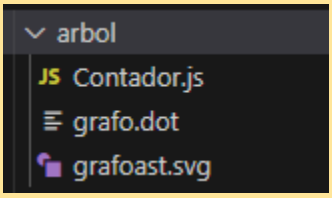
En la carpeta instrucciones contamos con diversidad de instrucciones como los if, case, while, for, creación de tablas, entre otros, todas estas estructuras hacen funcionamiento de como va la instrucción en la grmatica para que se pueda dar un resultado correcto de lo que se debería esperar en una ejecución del lenguaje SQL.



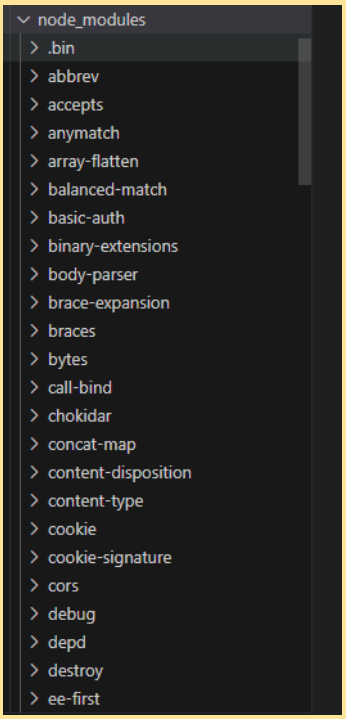
En la carpeta tabla de símbolos podemos encontrar diferentes clases de objetos, asi como la clase de entorno donde se crean las tablas, se guardan las variables, funciones, métodos etc., y se utilizan diferentes funciones para al utilización de estas.

Aquí vemos algunos archivos que se encuentran en estas carptas:

En la carpeta árbol podemos encontrar los reportes del árbol ast.



Saliendo de estas carpetas podemos encontrar la carpeta node_modules estos archivos son creados por defecto cuando se crea la api.


node_modules
.bin
abbrev
accepts
anymatch
array-flatten
balanced-match
basic-auth
binary-extensions
body-parser
brace-expansion
braces
bytes
call-bind
chokidar
concat-map
content-disposition
content-type
cookie
cookie-signature
cors
debug
depd
destroy
ee-first

Utilización de codigo:

Empezando por expresiones regulares:

Utilice la expresión regular para la fecha

http://w3.unpocodetodo.info/utiles/regex-ejemplos.php?type=fechas

Para realizar el árbol AST me guie de dos repositorios:

https://github.com/Mocta-996/Laboratorio-OLC1-C-1S2023

y el del auxiliar del curso

https://github.com/AlexIngGuerra/OLC1-2S2023

Para el archivo de grmatica y el parser me guie del repositorio del aux entre otras cosas:

Clases: gramática.jison , parser.json , controller , expresión: aritmética , dato , id ; instrucción : mostrar ; tablasimbolos : entorno, funciones , símbolo; instrucción.js

Funciones: ejecutar , getast;

https://github.com/AlexIngGuerra/OLC1-2S2023

Diagrama general de la aplicación:

Muestra el sistema de organización que se utilizo para desarrollar este sistema en una estructura UML de clases con ello podemos validar la fácil adaptación en caso de ser requerido a un nuevo sistema de lenguaje.