UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
LENGUAJES FORMALES Y DE PROGRAMACIÓN
SEGUNDO SEMESTRE

MANUAL

-

TÉCNICO

TABLA DE CONTENIDO

DATOS DEL DESARROLLADOR……1.

INTERFAZGRÁFICA……2-5.

LICENCIA DEL DESARROLLADOR

CHRISTOPHER IVÁN MONTERROSO ALEGRIA.

CARNÉ:201902363

ESTUDIANTE INGENIERIA EN CIENCIAS Y SISTEMAS.

DISEÑO FRONTEND:

PARA EL FRONTEND SE HA USADO LA HERRAMIENTE PARA FRONTEND DE NODE, VITE JS.

SE HA INTEGRADO VITE JS CON REACT JS PARA EL DISEÑO DEL FRONTEND.

VITE.JS

VITE.JS ES UN ENTORNO DE DESARROLLO ULTRARRÁPIDO PARA LA CONSTRUCCIÓN DE APLICACIONES WEB MODERNAS. DESTACA POR SU VELOCIDAD Y EFICIENCIA, LO QUE LO CONVIERTE EN UNA HERRAMIENTA IDEAL PARA EL DESARROLLO DE PROYECTOS FRONTEND. VITE.JS UTILIZA UNA ARQUITECTURA DE CONSTRUCCIÓN BASADA EN ESM (ECMASCRIPT MODULES) QUE PERMITE UNA CARGA INSTANTÁNEA Y UN TIEMPO DE COMPILACIÓN MÍNIMO. ESTO SE TRADUCE EN UNA EXPERIENCIA DE DESARROLLO FLUIDA, INCLUSO PARA APLICACIONES DE GRAN ENVERGADURA.

REACT

REACT ES UNA BIBLIOTECA DE JAVASCRIPT DESARROLLADA POR FACEBOOK QUE SE UTILIZA PARA CONSTRUIR INTERFACES DE USUARIO INTERACTIVAS Y REUTILIZABLES. SE CENTRA EN LA CREACIÓN DE COMPONENTES, PEQUEÑAS UNIDADES DE INTERFAZ DE USUARIO QUE PUEDEN COMBINARSE PARA CONSTRUIR INTERFACES COMPLEJAS. REACT UTILIZA UN PARADIGMA DE PROGRAMACIÓN LLAMADO "PROGRAMACIÓN DECLARATIVA", LO QUE SIGNIFICA QUE LOS DESARROLLADORES DESCRIBEN CÓMO DEBERÍA VERSE LA INTERFAZ EN UN ESTADO DADO Y REACT SE ENCARGA DE ACTUALIZARLA AUTOMÁTICAMENTE CUANDO LOS DATOS CAMBIAN. ESTO FACILITA LA CONSTRUCCIÓN DE APLICACIONES ESCALABLES Y MANTENIBLES.

INTEGRACIÓN DE VITE.JS Y REACT

VITE.JS SE INTEGRA SIN PROBLEMAS CON REACT, PROPORCIONANDO UN ENTORNO DE DESARROLLO ALTAMENTE EFICIENTE PARA PROYECTOS REACT. AL UTILIZAR VITE.JS CON REACT, LOS DESARROLLADORES PUEDEN APROVECHAR LA VELOCIDAD DE VITE.JS Y LA POTENCIA DE REACT PARA CREAR APLICACIONES WEB MODERNAS Y ATRACTIVAS DE MANERA EFICIENTE. LA CONFIGURACIÓN INICIAL ES SENCILLA, Y VITE.JS FACILITA LA IMPORTACIÓN DE DEPENDENCIAS EXTERNAS, LO QUE PERMITE A LOS DESARROLLADORES CENTRARSE EN LA CREACIÓN DE COMPONENTES Y LA LÓGICA DE LA APLICACIÓN.

EXPLICACIÓN DE APP.JSX

1.IMPORTS:

- EL CÓDIGO COMIENZA CON UNA SERIE DE IMPORTACIONES DE MÓDULOS DE REACT Y OTRAS DEPENDENCIAS, COMO **USESTATE**, **USEREF** DE REACT, Y COMPONENTES DE UNA LIBRERÍA LLAMADA ACEEDITOR QUE PROPORCIONA UN EDITOR DE TEXTO AVANZADO.

2.ESTADO Y VARIABLES:

- SE UTILIZAN LOS HOOKS **USESTATE** PARA CREAR DIFERENTES ESTADOS EN EL COMPONENTE, COMO **EDITORCONTENT**, **EXECUTIONRESULT**, **LEXERERRORS**,

SYNTAXERRORS, **SEMANTICERRORS**, **SYMBOLS** Y **SHOWTABLES**. ESTOS ESTADOS SERÁN UTILIZADOS PARA GESTIONAR LA INFORMACIÓN QUE SE MUESTRA Y SE ACTUALIZA EN LA INTERFAZ DE USUARIO.

3.FUNCIONES:

- **HANDLEFILEBUTTONCLICK**: ESTA FUNCIÓN ES LLAMADA CUANDO SE HACE CLIC EN UN BOTÓN PARA ABRIR UN ARCHIVO. UTILIZA LA REFERENCIA **FILEINPUTREF** PARA ABRIR UN CUADRO DE DIÁLOGO DE SELECCIÓN DE ARCHIVOS.
- **HANDLEFILEINPUTCHANGE**: ESTA FUNCIÓN MANEJA EL EVENTO CUANDO EL USUARIO SELECCIONA UN ARCHIVO. LEE EL CONTENIDO DEL ARCHIVO Y LO ESTABLECE COMO EL CONTENIDO DEL EDITOR.
- **HANDLESAVEBUTTONCLICK**: CREA UN ARCHIVO DE TEXTO A PARTIR DEL CONTENIDO ACTUAL DEL EDITOR Y LO DESCARGA.
- **HANDLERUNBUTTONCLICK**: SE EJECUTA CUANDO EL USUARIO HACE CLIC EN EL BOTÓN "EJECUTAR". REALIZA UNA SOLICITUD POST A **HTTP://LOCALHOST:8080/ANALYZE** CON EL CONTENIDO DEL EDITOR. LUEGO, ACTUALIZA LOS ESTADOS CON LOS RESULTADOS DEVUELTOS POR EL SERVIDOR.
- **READFILEASYNC**: ES UNA FUNCIÓN AUXILIAR QUE DEVUELVE UNA PROMESA QUE RESUELVE CON EL CONTENIDO DE UN ARCHIVO LEÍDO COMO TEXTO.
- **SHOWTABLESOK**: CAMBIA EL ESTADO **SHOWTABLES** PARA MOSTRAR LAS TABLAS DE RESULTADOS.

4.RENDERIZADO:

- LA FUNCIÓN **APP** DEVUELVE JSX QUE REPRESENTA LA INTERFAZ DE USUARIO. MUESTRA UN ENCABEZADO, UN EDITOR DE TEXTO (USANDO ACEEDITOR), BOTONES PARA REALIZAR VARIAS ACCIONES (COMO ABRIR, GUARDAR, EJECUTAR, ETC.), UNA CAJA DE TEXTO DE SOLO LECTURA PARA MOSTRAR EL RESULTADO DE LA EJECUCIÓN, Y COMPONENTES COMO **TABLESYMBOL** Y **TABLEERROR** PARA MOSTRAR TABLAS DE SÍMBOLOS Y ERRORES.

DISEÑO BACKEND:

1. **GOLANG (GO)**:
    - GO ES UN LENGUAJE DE PROGRAMACIÓN DE CÓDIGO ABIERTO DESARROLLADO POR GOOGLE. ES CONOCIDO POR SU SIMPLICIDAD, EFICIENCIA Y POTENCIA EN EL DESARROLLO DE SOFTWARE. GO ESTÁ DISEÑADO PARA SER RÁPIDO EN LA EJECUCIÓN Y FÁCIL DE ESCRIBIR Y MANTENER.
    - **REQUERIMIENTOS**:
        - PUEDES DESCARGAR E INSTALAR GO DESDE EL SITIO WEB OFICIAL: HTTPS://GOLANG.ORG/
        - ASEGÚRATE DE CONFIGURAR CORRECTAMENTE TU GOPATH Y AÑADIR EL DIRECTORIO **BIN** AL PATH PARA PODER EJECUTAR COMANDOS DE GO.
2. **ANTLR4**:
    - ANTLR (ANOTHER TOOL FOR LANGUAGE RECOGNITION) ES UNA HERRAMIENTA DE GENERACIÓN DE ANALIZADORES LÉXICOS Y SINTÁCTICOS. PERMITE DEFINIR GRAMÁTICAS FORMALES PARA DIFERENTES LENGUAJES Y GENERA CÓDIGO FUENTE QUE PUEDE SER UTILIZADO PARA ANALIZAR Y PROCESAR TEXTO QUE SIGUE ESAS GRAMÁTICAS.
    - **REQUERIMIENTOS**:
        - PUEDES DESCARGAR ANTLR4 DESDE EL SITIO WEB OFICIAL: HTTPS://WWW.ANTLR.ORG/
        - DEPENDIENDO DE TU SISTEMA OPERATIVO, LA INSTALACIÓN PUEDE VARIAR. POR LO GENERAL, IMPLICA DESCARGAR UN ARCHIVO JAR O USAR UN GESTOR DE PAQUETES ESPECÍFICO.

3. FIBER (GO FIBER):

- FIBER ES UN FRAMEWORK WEB LIGERO Y RÁPIDO PARA GO. ESTÁ DISEÑADO PARA SER FÁCIL DE USAR Y PROPORCIONA UN ALTO RENDIMIENTO PARA LA CONSTRUCCIÓN DE APLICACIONES WEB Y API RESTFUL.
- **REQUERIMIENTOS**:
    - PUEDES OBTENER FIBER UTILIZANDO EL COMANDO **GO GET**:
    - GO GET -U GITHUB.COM/GOFIBER/FIBER/V2

4. COMANDO PARA GENERACIÓN DE (ANTRL):

```
antlr4 -Dlanguage=Go -o parser -package parser -visitor *.g4
```

FUNCIONES PARA LA API.

```
app := fiber.New()
app.Use(cors.New())
app.Post("/analyze", func(c *fiber.Ctx) error {
    SemanticErrors = nil
    LexerErrors = nil
    SyntaxErrors = nil
    Symbols = nil
    prog := string(c.Body())

    input := antlr.NewInputStream(prog)
    lexer := parser.NewControlLexer(input)
    getLexerErrors := &LexerError{}
    lexer.AddErrorListener(getLexerErrors)
    tokens := antlr.NewCommonTokenStream(lexer, 0)
    getSyntaxsErrors := &SyntaxErrs{}
    p := parser.NewControlParser(tokens)
    p.RemoveErrorListeners()
    p.AddErrorListener(getSyntaxsErrors)
    p.BuildParseTrees = true
    tree := p.Prog()
    eval := Visitor{memory: make(map[string]Value)}
    eval.Visit(tree)
    graph.CreateGraph(prog)
    return c.JSON(fiber.Map{
        "prints":         eval.outputBuilder.String(),
        "LexerErrors":    LexerErrors,
        "SyntaxErrors":   SyntaxErrors,
        "SemanticErrors": SemanticErrors,
        "Symbols":        Symbols,
    })

})
```

FUNCIÓN POST ("/ANALYZE"):

- ESTA FUNCIÓN SE MANEJA CUANDO SE REALIZA UNA SOLICITUD HTTP POST A LA RUTA "/ANALYZE" DE LA APLICACIÓN.
- CUANDO SE RECIBE UNA SOLICITUD POST EN ESTA RUTA, PRIMERO SE REINICIAN LAS LISTAS DE ERRORES Y SÍMBOLOS (SEMANTICERRORS, LEXERERRORS, SYNTAXERRORS, SYMBOLS) PARA ASEGURARSE DE QUE ESTÉN VACÍAS.
- LUEGO, SE OBTIENE EL PROGRAMA FUENTE DEL CUERPO DE LA SOLICITUD HTTP.
- SE CREA UN ANALIZADOR LÉXICO Y SINTÁCTICO UTILIZANDO ANTLR4 PARA PROCESAR EL PROGRAMA FUENTE.
- SE DEFINE UN OBJETO VISITOR PERSONALIZADO PARA RECORRER EL ÁRBOL SINTÁCTICO GENERADO POR ANTLR4 Y EJECUTAR LA LÓGICA DE ANÁLISIS DEL PROGRAMA.
- SE CREA UN GRÁFICO (ARBOL CST) UTILIZANDO UNA FUNCIÓN CREATEGRAPH.
- FINALMENTE, SE RESPONDE A LA SOLICITUD POST CON UN JSON QUE INCLUYE INFORMACIÓN SOBRE LA EJECUCIÓN DEL PROGRAMA, ERRORES LÉXICOS, ERRORES SINTÁCTICOS, ERRORES SEMÁNTICOS Y SÍMBOLOS.

```go
app.Get("/graph", func(c *fiber.Ctx) error {
    // Lee el archivo SVG
    svgContent, err := ioutil.ReadFile("cst_graph_tree.svg")
    if err != nil {
        return c.Status(fiber.StatusInternalServerError).SendString("Error al leer el archivo SVG")
    }

    fmt.Println("SVG enviado...")
    return c.Type("svg").Send(svgContent)
})
```

Función GET ("/graph"):

- Esta función se maneja cuando se realiza una solicitud HTTP GET a la ruta "/graph" de la aplicación.
- Cuando se recibe una solicitud GET en esta ruta, la función lee el contenido de un archivo SVG llamado "cst_graph_tree.svg".
- Si se puede leer el archivo correctamente, lo envía como respuesta al cliente. Esto permite que el cliente obtenga el contenido SVG del gráfico para su visualización.