

Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Auxiliar: Maynor Pilo Tuy

Organización de Lenguajes y Compiladores 1

Proyecto 2 Typewise

Lesther Kevin Federico López Miculax

Carnet: 202110897

03/05/2023

DATOS DE DESARROLLADOR

Nombre del desarrollador: Lesther Kevin Federico López Miculax

Correo electrónico: lestherlopez64@gmail.com

Nacionalidad: guatemalteca

Carné universitario: 202110897

DETALLES DE DESARROLLO

LENGUAJE DE PROGRAMACIÓN

El proyecto realizado denominado con el nombre de Typewise, se logró gracias a diversos lenguajes de programación, los cuales destaca principalmente TypeScript, el cual es un lenguaje de programación libre y de código abierto que se basa principalmente en otro lenguaje de programación, el cual es JavaScript. De este modo, una característica importante de TypeScript es que añade características adicionales a JavaScript, como el tipado estático, clases, interfaces y módulos, los que lograron que se pudiera realizar el compilador del proyecto, facilitando el proceso en gran escala. Por lo que, en conclusión, TypeScript fue de gran ayuda y permitió que el proyecto fuera mas accesible de desarrollar sin mayores complicaciones.

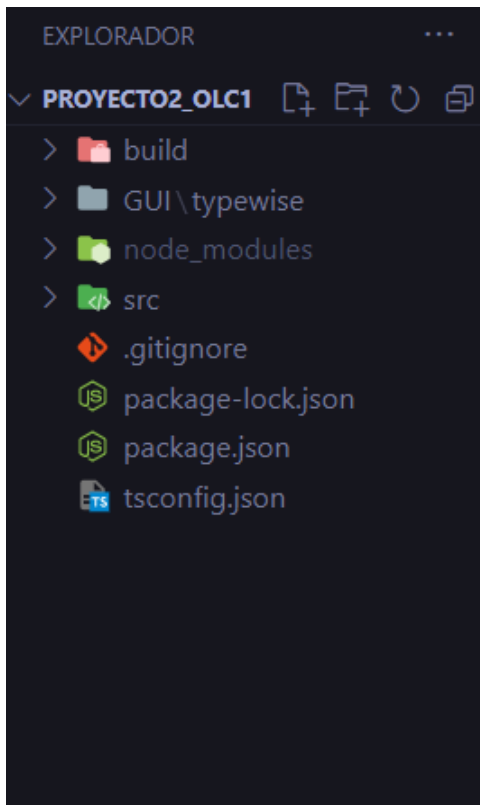
Por otro lado, también se hizo uso de lo que es la base de TypeScript, es decir, JavaScript, el cual es un lenguaje de programación de carácter interpretado y orientado a objetos que se utiliza principalmente en el desarrollo de aplicaciones web del lado del cliente y del servidor, por lo que para el frontend y backend de este proyecto se hizo uso de esta tecnología. Javascript es en lenguaje compatible con múltiples paradigmas de programación,

incluyendo programación funcional y programación orientada a objetos. Además, JS nos brindó el acceso a diferentes librerías como le fue en este caso React, la cual se hablará más adelante que fueron de gran utilidad e hicieron posible la implementación de la aplicación.



VISUAL STUDIO CODE

para poder desarrollar los lenguajes de programación se utilizó uno de los editores de texto más populares el cual es denominado con el nombre Visual Studio Code para poder crear los distintos archivos con extensión .ts y js, logrando así poder administrar diversas carpetas correspondientes a las vistas y funciones de la aplicación, tanto del backend y frontend.



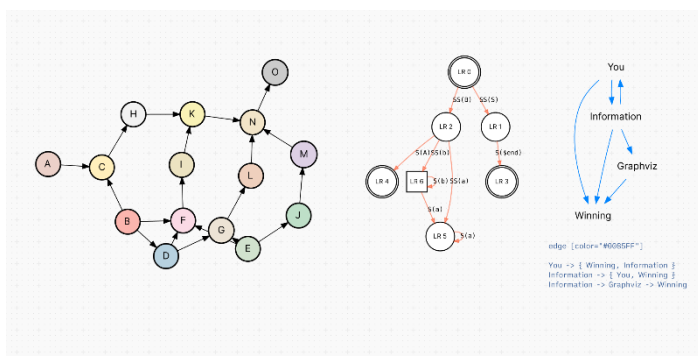
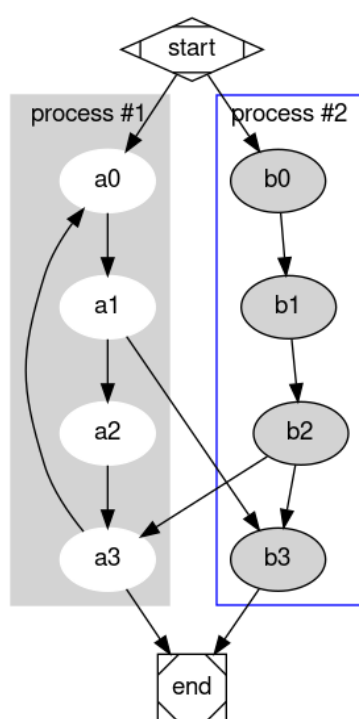
REACT

El principal causante del FronteEnd o Interfaz Gráfica de la aplicación TypeWise fue React, el cual a lo largo de los años se ha establecido como una herramienta fundamental y es una biblioteca de JavaScript de código abierto que se utiliza para construir interfaces de usuario para aplicaciones web y móviles. Fue desarrollada por Facebook y es utilizada por grandes empresas debido a su flexibilidad y accesibilidad., de este modo, en este proyecto se realizó un proyecto rápido y maquetado de react con el siguiente comando.

```
npx create-react-app nombreapp
```

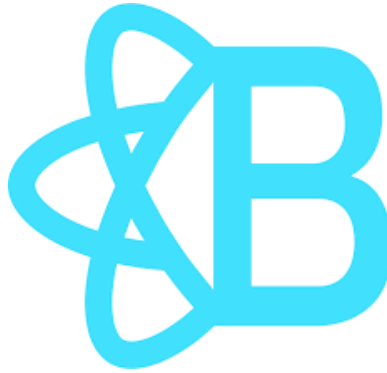
Además, react también cuenta con las opciones de agregarle librerías para poder tener una mejor funcionalidad e interfaz, las cuales se profundizaran a continuación.

Se utilizó la herramienta Graphviz para generar los reportes, como el árbol de análisis sintáctico, tablas de transiciones, tablas de siguientes y autómatas. Para lograr esto, se crearon archivos con extensión DOT que contenían información para generar las gráficas. DOT es un lenguaje de descripción de gráficos que permite crear gráficas y diagramas. Una vez generados los archivos DOT, se convirtieron a archivos JPG para poder visualizarlos.



React Bootstrap es otra fundamental pieza de esta maquina denominada TypeWise, el cual es una biblioteca de alta tecnología de componentes de interfaz de usuario de código abierto construida en React, de este modo, esta herramienta proporciona una gran cantidad de componentes preconstruidos que pueden utilizarse para construir rápidamente interfaces de usuario con una apariencia y funcionalidad consistentes, incluyendo botones, menús

desplegables, tablas, formularios y más. Es a destacar que para este proyecto se usaron botones, modales para mostrar gráficas, entre otras herramientas.



JISON

Una principal herramienta que logro que nuestro compilador funcionara y lograra reconocer lo que se le ingresaba es la tecnología de JISON, la cual genera un analizado sintáctico para cualquier lenguaje de programación, en este caso es el del lenguaje establecido que admite TypeWise. Jison se basa en el popular generador de analizadores sintácticos Bison y utiliza una combinación de técnicas de análisis léxico y sintáctico para producir un parser eficiente en JavaScript. Además, es importante destacar que JISON genera el parser de manera automática para poder analizar el texto de entrada.

```

3  /* Definición Léxica */
4  %lex
5
6  %options case-insensitive
7  %x string
8
9  int          (?:[0-9]|[1-9][0-9]+)
10
11 frac         (?:\.[0-9]+)
12
13 %%
14
15 // simbolos reservados
16 ";"          return 'PTCOMA';
17 "("          return 'PARIZQ';
18 ")"          return 'PARDER';
19 "."          return 'PUNTO';
20 ":"          return 'DOSPUNTOS';
21 ","          return 'COMA';

```

NODE

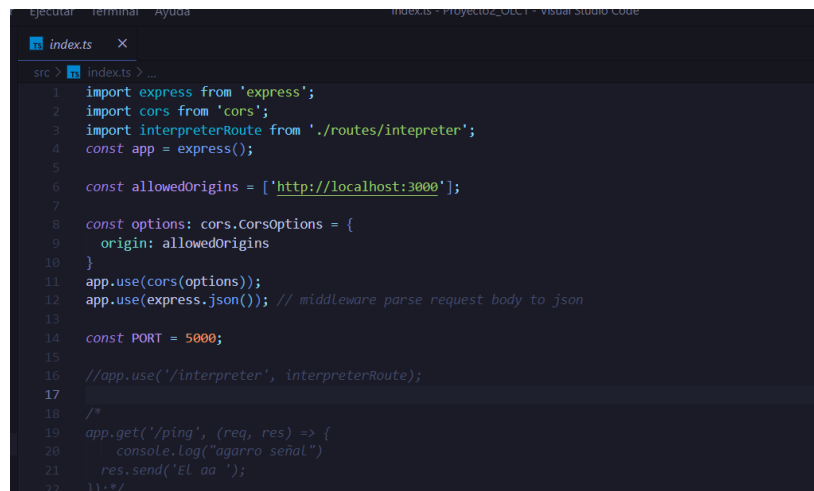
Luego, de conocer cuáles fueron las tecnologías que corresponden al frontend, también es crucial conocer como fue formado el back, el cual es el lado del servidor. Por lo que, para esto se utilizó la tecnología de Node.js, la cual es un entorno de tiempo de ejecución basado en el lenguaje de programación JavaScript. Node.js utiliza el motor de JavaScript V8 de Google para compilar y ejecutar el código JavaScript de manera eficiente. Además, Node.js tiene una gran comunidad de desarrolladores que contribuyen con paquetes y herramientas de código abierto para simplificar el desarrollo de aplicaciones en Node.js.



LOGICA DE PROGRAMA

INDEX

El index del programa y del backend contiene dentro de el un servidor Express que utiliza el módulo de cors y una ruta llamada "interpreter" para manejar solicitudes. Primero, se importan los módulos necesarios de Express y cors, así como la ruta "interpreter". Luego, se crea una instancia de la aplicación Express. Además, dentro del Index se define que el servidor se ejecuta en el puerto 5000 y utiliza la ruta "/interpreter" para manejar las solicitudes entrantes. Finalmente, se imprime un mensaje en la consola indicando que el servidor está en funcionamiento y escuchando en el puerto especificado. Finalmente, A continuación, se utiliza el método use() para aplicar el middleware express.json(), que analiza el cuerpo de la solicitud en formato JSON.



```
1 import express from 'express';
2 import cors from 'cors';
3 import interpreterRoute from './routes/intepreter';
4 const app = express();
5
6 const allowedOrigins = ['http://localhost:3000'];
7
8 const options: cors.CorsOptions = {
9   origin: allowedOrigins
10 }
11 app.use(cors(options));
12 app.use(express.json()); // middleware parse request body to json
13
14 const PORT = 5000;
15
16 //app.use('/interpreter', interpreterRoute);
17
18 /*
19 app.get('/ping', (req, res) => {
20   console.log("agarro señal")
21   res.send('El aa ');
22 });*/
```

RUTA INTERPRETAR

El archivo de ruta interpretar es uno de ruta en Express para la aplicación del intérprete. Importa el módulo express para crear un objeto de router que maneja las

solicitudes HTTP GET y POST. Sin embargo, en esta aplicación se hizo uso únicamente del POST.

```
src > routes > interpreter.ts > default
1 import express from 'express';
2 import { interpreteController } from '../controllers/interpretecontroller';
3 const router = express.Router();
4
5 // controlador
6 router.get('/ping', interpreteController.pong);
7
8 // interpretar código fuente
9 router.post('/interpretar', interpreteController.interpretar);
10 export default router;
```

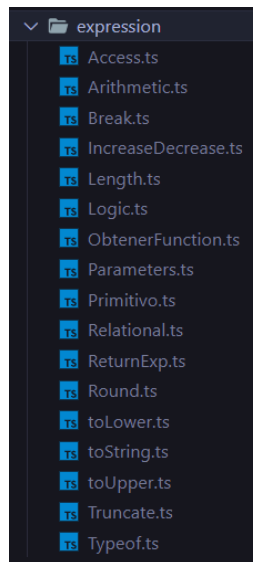
EXPRESION

Para las expresiones se usó un código con una clase abstracta llamada "Expression", la cual tiene dos propiedades públicas llamadas "line" y "column" que representan la línea y columna en la que se encuentra la expresión. Esta clase se utiliza posteriormente para poder realizar las operaciones o funciones que tiene cada uno de las expresiones que utiliza el lenguaje.

```
controllers > interpreter > abstract > Expression.ts > Expression
import { Return } from './Return';
import { Environment } from './Environment';
export abstract class Expression {
  public line: number;
  public column: number;
  constructor(line: number, column: number) {
    this.line = line
    this.column = column
  }

  public abstract execute(env: Environment): Return;
  public abstract AST(): {rama: string, nodo: string};
}
```

Por otro lado, cada una de las expresiones del programa y que reconoce se les creo una clase por aparte donde dentro de ellas contiene la funcionalidad para que cada uno de ellos e ejecute de buena forma.



Un claro ejemplo es lo que contiene el archivo Round, el cual dentro de el contiene un código que define la clase Round que hereda de Expression. Round recibe otra expresión como parámetro y redondea su valor numérico hacia arriba o hacia abajo dependiendo del decimal que tenga. Si la expresión recibida es de tipo entero o doble, se realiza el redondeo y se devuelve el valor redondeado como entero. Si la expresión no es numérica, se devuelve un valor nulo. El método execute se encarga de realizar esta lógica de redondeo y devuelve el valor resultante. Este código forma parte de una implementación de un lenguaje de programación y se utiliza para realizar operaciones matemáticas en el mismo.

```

export class Round extends Expression {
  constructor(
    private expression: Expression,
    line: number,
    column: number
  ) {
    super(line, column);
  }

  public execute(env: Environment): Return {
    const valor = this.expression.execute(env); // value and type

    if(valor.type == Type.INT || valor.type == Type.DOUBLE){
      const entero = Math.floor(valor.value);
      const decimal = valor.value - entero;
      if (decimal >= 0.5) {
        // si el decimal es mayor o igual que 0.5, redondeamos al número superior
        return { value: Math.ceil(valor.value), type: Type.INT };
      } else {
        // si el decimal es menor que 0.5, redondeamos al número inferior
        return { value: Math.floor(valor.value), type: Type.INT };
      }
    }
  }
}

```

Del mismo modo, las diferentes expresiones se trabajaron de igual manera logrando que se reconozcan expresiones como obtener dunciones, acceder a variables, operaciones aritméticas, entre otras.

INSTRUCCIONES

Para las instrucciones se realizó un proceso similar a la de las expresiones la cual se creo una clase abstracta llamada "Instruction" que tiene dos propiedades "line" y "column" y un constructor que inicializa estas propiedades con valores dados. Además, la clase tiene un método abstracto llamado "execute" que toma dos parámetros: "env", que representa el entorno actual, y "id", que representa el identificador actual. El propósito de esta clase es proporcionar una base para otras clases que implementen la lógica específica de una instrucción en particular. Al ser abstracta, esta clase no puede ser instanciada directamente, sino que debe ser extendida por otras clases que proporcionen la implementación concreta del método "execute".

Por lo que, se creó una carpeta denominada Insatructions la cual dentro de ella tiene varios archivos donde cada uno de ellos tiene una clase que permite ejecutar cada una de las instrucciones un claro ejemplo de esto es el del bucle while.

```
export class While extends Instruction{  
  
  exp: Expression;  
  sentencias : Instruction;  
  
  constructor(exp: Expression, sentencias : Instruction, linea: number, columna: number){  
    super(linea, columna);  
    this.exp = exp;  
    this.sentencias = sentencias;  
  }  
  
  public execute(env: Environment) {  
    let condicional = this.exp.execute(env);  
    //creacion de entorno  
  
    let environment_while = new Environment(env, "while");  
    while(condicional.value){  
      let elemento = this.sentencias.execute(environment_while, "while");  
      condicional = this.exp.execute(environment_while);  
    }  
  }  
}
```

El código de la clase while extiende de la clase abstracta Instruction. Representa la estructura de control while en el lenguaje de programación que se está desarrollando. El método execute es el encargado de ejecutar las instrucciones en el orden adecuado. Primero, se evalúa la condición (exp) utilizando el método execute de la clase Expression, pasándole el entorno actual (env). Luego, se crea un nuevo entorno (environment_while) para las instrucciones dentro del while. Por consiguiente, dentro de un ciclo while, se ejecutan las instrucciones (sentencias) en cada iteración mientras la condición (condicional.value) sea verdadera. Si se encuentra una instrucción return, se detiene la ejecución del ciclo y se devuelve el valor del return.