Manual Técnico

JavaBridge: Traductor Java a Python

Angel Raúl Herrera Chilel
Carnet: 202402587
Sección: B
25 de Octubre de 2025

Índice

Índice

Índice **1**

1. Introducción **2**

2. Repositorio **2**

3. Requerimientos del Sistema (Desarrollo) **2**
3.1. Reporte Técnico 2
3.2. Tecnologías y Dependencias 2

4. Explicación Detallada de Funciones/Métodos **2**
4.1. Backend (Carpeta `/backend`) 2
4.1.1. main.js 3
4.1.2. Lexer.js (Clase Lexer) 3
4.1.3. Parser.js (Clase Parser) 3
4.1.4. services/pythonRunner.js 4
4.2. Frontend (Carpeta `/frontend`) 4
4.2.1. main.js (Cliente) 4

5. Conclusiones **5**

1 Introducción

El presente manual técnico describe la estructura y funcionamiento interno del sistema *JavaBridge*, una aplicación cliente-servidor desarrollada en JavaScript para traducir un subconjunto de Java a Python. Este documento está dirigido a desarrolladores que necesiten comprender el código fuente, la arquitectura y los métodos de análisis. El sistema se compone de un **backend** en Node.js/Express que expone una API REST para el análisis, y un **frontend** en JavaScript puro (servido con Vite) que consume dicha API y presenta la interfaz gráfica.

2 Repositorio

Repositorio en GitHub: Enlace al código fuente

3 Requerimientos del Sistema (Desarrollo)

3.1 Reporte Técnico

El programa fue desarrollado y probado en el siguiente entorno:

- **Sistema Operativo:** ArchLinux (Kernel 6.x) / Windows 11 / macOS Sonoma
- **IDE utilizado:** Visual Studio Code
- **Runtime (Backend):** Node.js v22.x
- **Servidor (Frontend):** Vite v5.x
- **Navegador:** Google Chrome (v120+)

3.2 Tecnologías y Dependencias

- **Backend:** `express` (servidor API), `cors` (middleware).
- **Frontend:** `vite` (servidor de desarrollo).
- **Simulación:** `child_process` (módulo nativo de Node.js).

4 Explicación Detallada de Funciones/Métodos

A continuación, se describen los principales componentes del sistema.

4.1 Backend (Carpeta `/backend`)

El backend es responsable de todo el análisis y la simulación.

4.1.1. main.js

Punto de entrada del servidor.

- **Propósito:** Configura el servidor Express y define la API.
- **Funcionamiento:**
- Inicia un servidor Express en el puerto 4000.
- Define la ruta principal: `POST /analizar`.
- **Ruta `/analizar`:** Recibe el `{ code }` en el body. Orquesta el proceso:
  1. Crea `new Lexer(code)` y llama a `lexer.analizar()`.
  2. Crea `new Parser(tokens)` y llama a `parser.analizar()`.
  3. Llama a `generarReporteHtml(...)` con los resultados.
  4. Llama a `simularPython(...)` con el código Python generado.
  5. Devuelve un JSON con: `{ tokens, lexicalErrors, syntaxErrors, pythonCode, reporteHtml, simulacion }`.

4.1.2. Lexer.js (Clase Lexer)

Analizador léxico manual (scanner).

- **Propósito:** Convertir el string de código Java en una lista de tokens.
- **Funcionamiento de métodos clave:**
- `analizar()`: Itera sobre el texto de entrada. Utiliza métodos auxiliares para identificar y generar tokens, omitiendo espacios en blanco.
- `comentarioLinea()` y `comentarioBloque()`: Identifican comentarios y los generan como tokens (`COMENTARIO_LINEA`, `COMENTARIO_BLOQUE`) para que el Parser pueda traducirlos, en lugar de descartarlos.
- `identificador()`: Reconoce palabras clave (de `ReservedWords`) e identificadores.
- `numero()`, `cadena()`, `caracter()`: Métodos para tokenizar literales, con manejo de errores (ej. cadenas sin cerrar).

4.1.3. Parser.js (Clase Parser)

Analizador sintáctico manual (tipo LL(1) / Descenso Recursivo).

- **Propósito:** Validar la estructura gramatical y generar el código Python.
- **Funcionamiento de métodos clave:**
- `programa()`: Punto de entrada. Valida la estructura obligatoria `public class ID { MAIN }`.
- `main()`: Valida la firma exacta de `public static void main(String[] args) { SENTENCIAS }`.

- `sentencias()`: Bucle que consume tokens y deriva a las reglas de sentencia apropiadas (declaración, if, for, while, print, comentarios) hasta encontrar una `LLAVE_DER`.
- `declaracion()`: Maneja `TIPO LISTA_VARS ;`. Traduce `int a;` a `a = 0` y `String b = "h";` a `b = "h"`.
- `forStmt()`: Traduce el `for` de Java a un `while` en Python, emitiendo la inicialización antes del bucle y la actualización (`i += 1`) al final del bloque `while`.
- `printStmt()`: Traduce `System.out.println(expr)` a `print(expr)`.
- `expresion()`: Método simplificado para construir la cadena de expresión. Maneja la concatenación con `str()` implícitamente si detecta strings.

4.1.4. services/pythonRunner.js

- **Propósito:** Ejecutar el código Python generado de forma segura.
- **Funcionamiento:**
- `simularPython(code)`: Utiliza el módulo `spawnSync` de `child_process` para ejecutar `python3 -c çode"`.
- Captura y devuelve `stdout` (salida estándar) y `stderr` (errores de ejecución) como un objeto.

4.2 Frontend (Carpeta `/frontend`)

El frontend es la interfaz de usuario que consume la API.

4.2.1. main.js (Cliente)

- **Propósito:** Manejar todos los eventos DOM y la comunicación con la API.
- **Funcionamiento de métodos clave:**
- `btnTraducir.addEventListener('click', ...)`: Función `async` que usa `fetch` para hacer `POST` al `API_URL` (`/analizar`) con el código Java. Al recibir el JSON, actualiza el `textContent` del editor Python o de la consola de errores.
- `btnReportes.addEventListener('click', ...)`: Abre una nueva ventana (`window.open()`) y usa `document.write()` para volcar el `reporteHtml` (recibido del JSON) en ella.
- `javaEditor.addEventListener('keydown', ...)`: Intercepta `e.key === 'Tab'`, previene la acción por defecto y usa `document.execCommand('insertText', ...)` para insertar dos espacios.
- `updateLineNumbers()`: Sincroniza el `div` de números de línea. Cuenta las líneas en el editor (`textContent.split('\n').length`) y genera el HTML para la barra lateral.

5 Conclusiones

El sistema *JavaBridge* implementa un traductor funcional utilizando una arquitectura cliente-servidor desacoplada. El análisis léxico y sintáctico manual en el backend de Node.js cumple con los requisitos del proyecto, y la interfaz de JavaScript puro proporciona una experiencia de usuario reactiva para la edición y simulación.