**Project Documentation: SNOBOL4 Web Service Wrapper**

**1. Objective**

The primary goal of this project was to make the legacy, command-line SNOBOL4 programming language accessible to modern web applications. The original SNOBOL4 tool runs as a standalone, compiled executable that accepts a source file and prints output to the standard console.

This project encapsulates that command-line tool within a Node.js web server, exposing its functionality through a simple REST API. This allows any JavaScript-based client (or any other application capable of making HTTP requests) to execute SNOBOL4 code and receive the results without needing the SNOBOL4 compiler or runtime installed locally.

**2. Architecture & Implementation**

The system is designed with a classic three-component architecture: a frontend client, a backend server (the wrapper), and the core SNOBOL4 engine.

**Data Flow:** [Browser Client] --(HTTP Request)--> [Node.js Server] --(Spawns Process)--> [SNOBOL4 Executable] [Browser Client] <-- (HTTP Response) -- [Node.js Server] <-- (Captures stdout/stderr)-- [SNOBOL4 Executable]

**Components:**

* **SNOBOL4 Executable (snobol4):** This is the pre-compiled Linux binary that we created. It is the core engine that executes SNOBOL4 code. The server treats it as a black box, simply running it as a command-line tool.
* **Backend Server (server.js):** This is the heart of the "wrapper". It's a lightweight server built using Node.js and the Express framework. Its responsibilities are:
  1. **Listen for API Calls:** It exposes an HTTP endpoint (POST /run-snobol) that waits for incoming requests.
  2. **Handle Requests:** When a request containing SNOBOL4 code arrives, the server writes this code to a temporary local file (e.g., temp\_12345.snb).
  3. **Process Management:** It uses Node.js's built-in child\_process.spawn() method to execute the snobol4 binary, passing the path to the temporary source file as an argument.
  4. **Capture Output:** It captures all data from the child process's standard output (stdout) and standard error (stderr).
  5. **Send Response:** Once the snobol4 process finishes, the server packages the captured output and error streams into a JSON object and sends it back to the client as an HTTP response.
  6. **Cleanup:** It deletes the temporary source file.
* **Frontend Client (index.html):** A simple HTML page that provides a user interface. It contains a <textarea> for users to write SNOBOL4 code and a <button>. The embedded client-side JavaScript uses the fetch API to send the contents of the textarea to the backend's /run-snobol endpoint and then displays the returned output or error on the page.

**3. API Usage Guide: How to Use from JavaScript**

To execute SNOBOL4 code, send an HTTP POST request to the server's primary endpoint.

**Endpoint:** POST /run-snobol

**Headers:**

* Content-Type: application/json

**Request Body:** The body must be a JSON object with a single key, code, which contains the SNOBOL4 source code as a string.

*Example JavaScript fetch request:*

JavaScript

async function executeSnobol(snobolCode) {

const endpoint = 'http://localhost:3000/run-snobol'; // Or your deployed server URL

try {

const response = await fetch(endpoint, {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ code: snobolCode })

});

if (!response.ok) {

throw new Error(`HTTP error! status: ${response.status}`);

}

const result = await response.json();

return result;

} catch (error) {

console.error("Failed to execute SNOBOL code:", error);

return { error: "Failed to connect to the server." };

}

}

// --- Usage Example ---

const myCode = " OUTPUT = 'This is a test from JS!' \nEND";

executeSnobol(myCode).then(result => {

console.log("SNOBOL4 Output:", result.output);

if (result.error) {

console.error("SNOBOL4 Error:", result.error);

}

console.log("Exit Code:", result.exitCode);

});

**Success Response (200 OK):** The server will respond with a JSON object containing the results of the execution.

*Example success response:*

JSON

{

"output": "This is a test from JS!\n",

"error": "",

"exitCode": 0

}

* output: (string) The captured standard output from the SNOBOL4 program.
* error: (string) The captured standard error. Will be empty on successful execution.
* exitCode: (number) The exit code of the SNOBOL4 process. 0 typically indicates success.

**4. Project Setup and Running Locally**

1. **Prerequisites:** Node.js, npm.
2. **File Structure:** Ensure the project directory contains:
   * snobol4 (the Linux executable)
   * server.js
   * index.html
   * package.json
3. **Installation:** From the project directory, run npm install to install Express.
4. **Run Server:** Execute node server.js.
5. **Access:** Open a web browser and navigate to http://localhost:3000.

**5. Deployment Notes (Render)**

To deploy this project to a service like Render:

1. Push the entire project to a GitHub repository.
2. **Crucially**, ensure the snobol4 file has execute permissions within the Git index by running git update-index --chmod=+x snobol4 before committing.
3. Connect your GitHub repository to Render as a new "Web Service".
4. Render will automatically detect the package.json file. Set the following commands:
   * **Build Command:** npm install
   * **Start Command:** node server.js