CS.CODE.IN

thesis-5
Online Web Compiler

Student: Syed Sania

Table of Content

1. Introduction

Overview

The compiler translates the source code into machine code. The aim of this thesis is to develop an online web compiler that allows users to write, compile, and run code in various programming languages directly from their web browser. This solution will support multiple languages, including C, C++, Java, Python, JavaScript, Ruby, and Bash.

Motivation

The need for an online web compiler arises from the increasing demand for accessible programming environments that can be used from any device with internet access. Such a tool is beneficial for educational purposes, quick prototyping, and collaborative coding.

Objectives

To build a user-friendly web interface for code input and execution.
To support multiple programming languages.
To ensure secure and efficient execution of user code.
To provide real-time feedback on code performance metrics.

2. System Design

Architecture Overview

The system follows a client-server architecture where the frontend interacts with the backend via REST APIs. The backend handles code compilation and execution, leveraging Node.js clusters for load balancing and efficiency.

Component Description

* Frontend: Provides a code editor, language selection, and displays output and errors.
* Backend: Processes compilation and execution requests, manages system resources, and ensures security.
* Middleware: Handles request parsing, system calls, and response formatting.

* Process Manager: Manages child processes for executing user code, ensuring isolation and resource limits.

Data Flow

1. User writes code and submits.
2. Frontend sends code to backend via API.
3. Backend middleware processes the request.
4. Code is compiled/executed in a controlled environment.
5. Output and metrics are sent back to the frontend.

3. System Architecture

a) Client Side Implementation
b) Server Side Implementation

The system architecture consists of two main components: the client-side and the server-side. The client-side is responsible for the user interface and interactions, while the server-side handles code compilation, execution, and result processing.
*User Interface Elements
*Form Submission
*Code Execution Response

a)Client-Side Implementation

User Interface Design

The user interface (UI) is designed using React and Ace Editor to provide a rich coding experience. The UI includes dropdowns for language and theme selection, a text area for code input, and fields for displaying output, errors, and performance metrics.

Key Components:

1. Language and Theme Selection: Dropdown menus for choosing the programming language and editor theme.
2. Code Editor: Integrated Ace Editor with support for multiple languages and themes.
3. Output and Error Display: Text areas for showing compilation results or error messages.
4. Performance Metrics: Fields for displaying time and memory usage of the executed code.

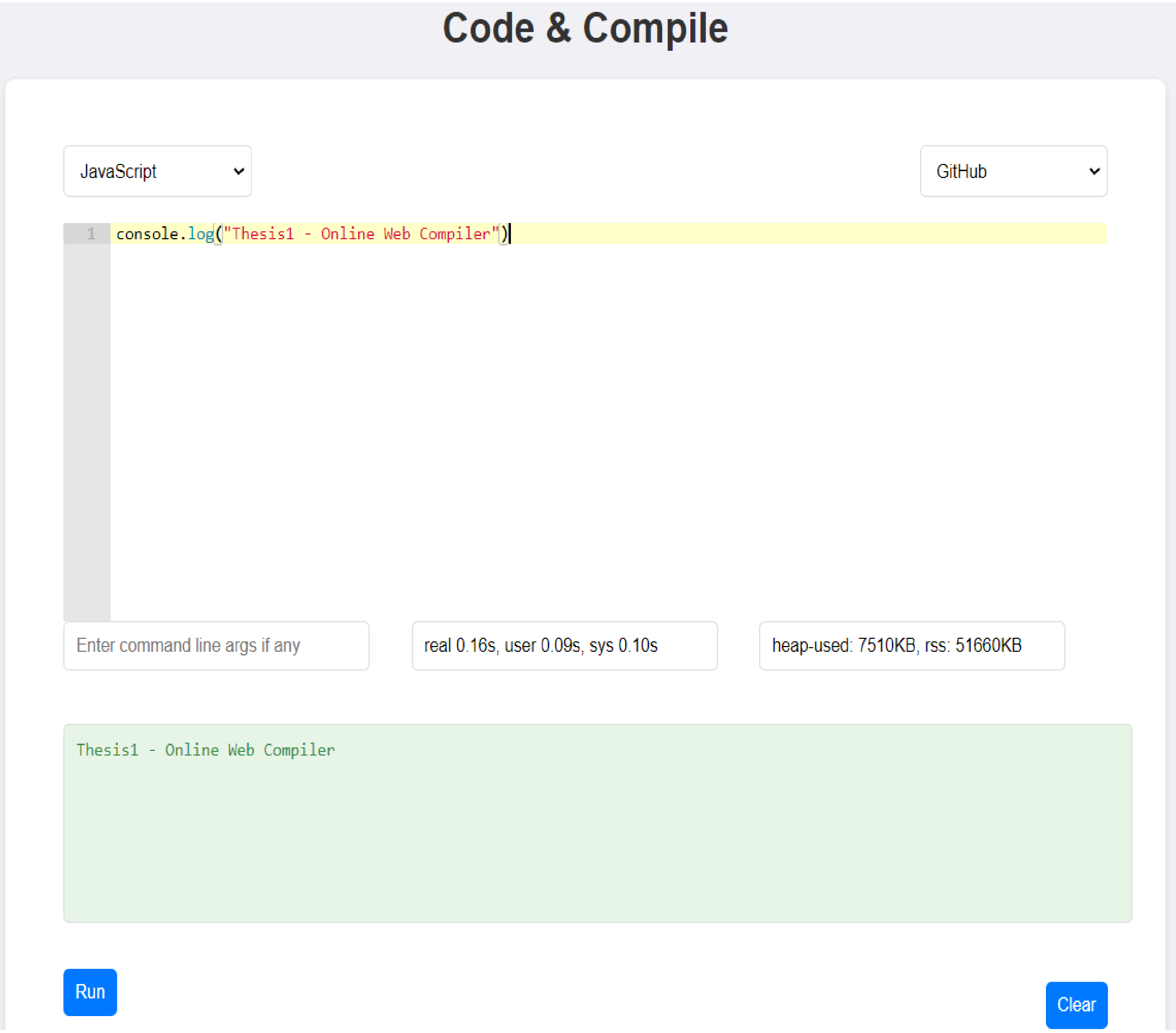
Code Editor Integration

Ace Editor is used for code entry, supporting syntax highlighting, autocompletion, and various themes. It is configured to handle different programming languages based on user selection.

Form Handling and Data Submission

The form includes fields for code input, language selection, and command-line arguments. Upon submission, the data is sent to the server using an Axios POST request. The client shows a loading indicator while waiting for the server response.

Below is a flowchart representing the client-side interactions:



```
console.log("Thesis1 - Online Web Compiler")
```

Enter command line args if any

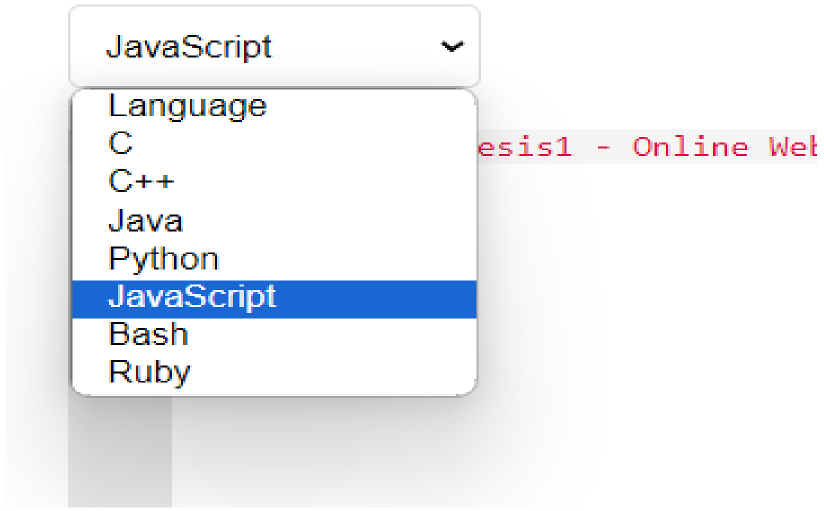
real 0.16s, user 0.09s, sys 0.10s

heap-used: 7510KB, rss: 51660KB

```
Thesis1 - Online Web Compiler
```

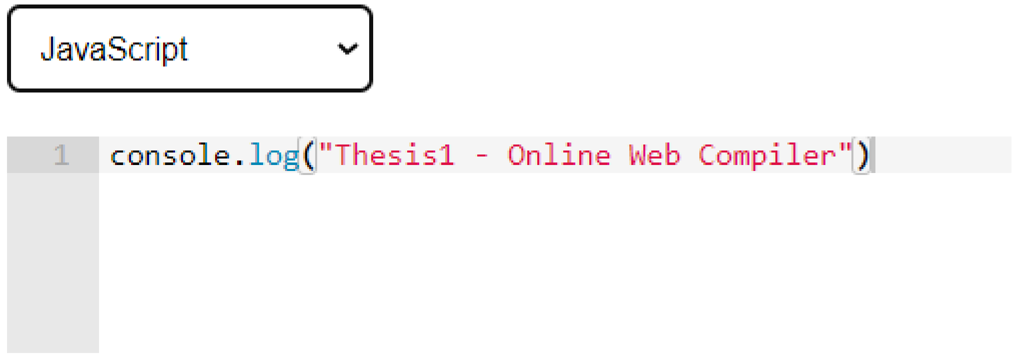
Step 1:
In Client side the user firsts makes a selection of the language which they wants to compile or interpret

Select the Language Using Select tag



```
esis1 - Online Wel
```

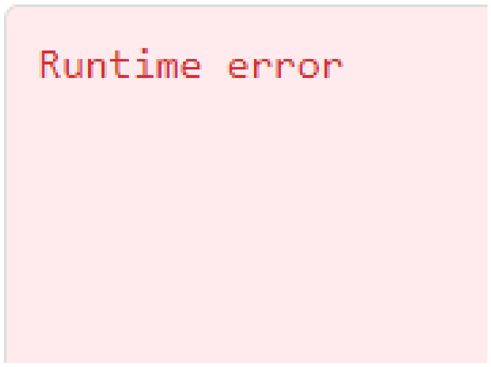
Step 2:
After writing the code



```
console.log("Thesis1 - Online Web Compiler")
```

The moment you click on Run Button its makes an API Call

If compilation fails, capture error and send response.



```
Runtime error
```

b)Server Side Implementation

API Endpoints

The server exposes API endpoints to handle code compilation and execution requests. Each endpoint corresponds to a specific programming language.

Middleware Functions

Middleware functions are used to process incoming requests, execute system calls for code compilation/execution, and handle responses.

Key Middleware Functions:

Language Middleware: Parses the request, sets up necessary arguments, and prepares the code for compilation/execution.
System Call Middleware: Executes the code using exec or execFile functions, captures performance metrics, and handles process termination.
Response Middleware: Sends the execution results, errors, and performance metrics back to the client.

Code Compilation and Execution

The server uses exec for interpreted languages and execFile for compiled languages. It handles code execution within specified resource limits .

Server-Side Flowchart

Start

1. Server is initialized and ready to accept requests.

Receive API Request

1. Server receives the POST request from the client with code, language, and arguments.

Language Middleware

1. Middleware processes the request based on the selected language.
2. Sets up necessary arguments for compilation or execution.

System Call Middleware

1. Middleware handles the system call for compiling (if necessary) and executing the code.
2. Captures performance metrics (execution time, memory usage).

Compilation (if required)

1. For compiled languages, compile the code using execFile.
2. If compilation fails, capture error and send response.

Execution

1. Execute the code using exec.
2. Capture output, errors, and performance metrics.

Process Termination

1. Ensure the process is terminated within the specified limits (timeout and memory).

Response Formation

1. Format the response with output, errors, performance metrics.
2. Send the response back to the client.

End

4.Implementation

Backend Development

1. Express.js Setup: Creating routes for different languages.
2. Child Processes: Using exec and execFile from child_process to handle code execution.
3. Cluster Management: Distributing load across CPU cores using Node.js clustering.

Frontend Development

1. React.js Setup: Creating components for the code editor, output display, and controls.
2. Ace Editor Integration: Integrating Ace Editor for a rich code editing experience.
3. State Management: Using React's state and hooks to manage application state.

Middleware Components

1. Request Handlers: Middleware to handle different programming languages, parsing request data.

System Call Middleware: Executing code and capturing performance metrics.
Response Middleware: Formatting and sending responses back to the client.

Process Management

1. Execution Timeouts: Limiting execution time to prevent infinite loops.
2. Memory Limits: Restricting memory usage to prevent resource exhaustion.
3. Security: Isolating execution environment to prevent security breaches.

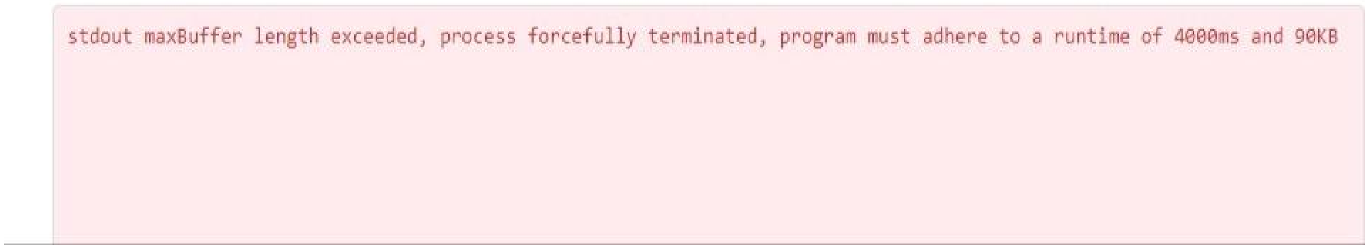
5. Binaries

For Generating Binaries we are using NPM Package pkg
Installation : sudo npm install pkg
1. Convert Whole Code to ES5 Modules
2. Remove type module from package.json
3. Add "bin": "app.js" in package.json
4. Remove nodemon from start script
5. pkg app.js
It creates 3 Binaries Window, Ubuntu and Mac

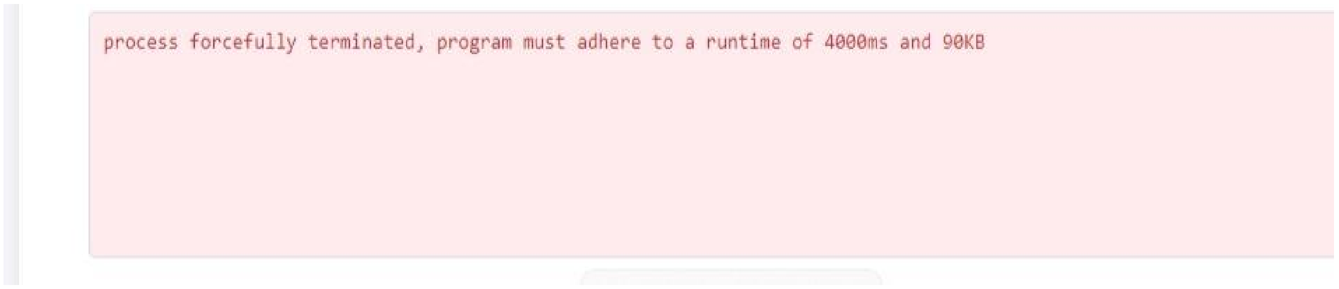
```
> pkg@5.8.0
> Targets not specified. Assuming:
  node18-linux-x64, node18-macos-x64, node18-win-x64
> Fetching base Node.js binaries to PKG_CACHE_PATH
  fetched-v18.5.0-linux-x64
  fetched-v18.5.0-macos-x64
  fetched-v18.5.0-win-x64
```

6.Error Handling

The project includes comprehensive error handling on both client and server sides. Errors are captured and displayed to the user, including compilation errors, runtime errors, and server-related issues.



```
stdout maxBuffer length exceeded, process forcefully terminated, program must adhere to a runtime of 4000ms and 90KB
```



```
process forcefully terminated, program must adhere to a runtime of 4000ms and 90KB
```

7.Security Considerations

Security measures are implemented to prevent unauthorized code execution and resource abuse:

*Input Validation: Ensures only valid code and arguments are processed.
*Resource Limits: Enforces strict limits on execution time and memory usage.
*Process Isolation: Runs code in isolated processes to prevent interference with the server.