VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CODEON

A Project Report

Submitted By:

Prakash Kumar 20BCE0080

Course Code:

CSE3002

Course Title:

Internet and Web Programming

Submitted To:

Dr. Rajkumar R.

SCOPE, VIT Vellore

Table of Contents

1. Abstract:

A code-compiler is a unique application that converts the source code of one programming language into machine code, bytecode, or another programming language. Usually, the source code is created in a high-level, readable language for humans, such Java or C++. An integrated development environment (IDE) or a code editor is used by the programmer to write the source code, which is then saved to one or more text files. The files are read, the code is examined, and it is then translated into a format appropriate for the target platform by a compiler that supports the source programming language.

2. Introduction:

I am going to build a code editor (CodeOn) that has the following features:

- A Code Editor (Monaco Editor) that powers VS Code too.
- It can compile code on a web app with standard input and output with support to over 40 programming languages.
- We can change the theme of the editor from a list of available themes.
- We can get information on the code executed (time taken by the code, memory used, status, and more)

3. Tech Stack:

For the project, I am going to use the following tech stack:

- React.js – For the front-end
- TailwindCSS – For styles
- Judge0 – For compiling and executing our code.
- RapidAPI – For quickly deploying Judge0 code.
- Monaco Editor – The code editor that powers the project.

4. Project Structure:

The project structure is fairly simple and easy to understand:

- **Components**: All the components / reusable code snippets live here (Example: CodeEditorWindow and Landing)
- **hooks**: All the custom hooks are present here. (using keypress hooks to compile our code using keyboard events)
- **lib**: All the library functions live here. (We'll create a function to define our theme here)
- **constants**: All the constants like languageOptions and customStyles for dropdowns will go here.
- **utils**: General utility functions to help maintain the code go here.

5. Flow of the Application:

- A user lands on the web application and can select their preferred programming languages (default is C++).
- Once the user is done writing their code, they can compile their code and see the output / results in the output window.
- They'll either see success or failure for their code snippets. Everything is visible in the code output window.
- The user can add custom inputs to their code snippets, and the judge (our online compiler) will take into account the custom input which the user supplies.
- The user can see relevant details of the code that was executed (Example: It took 5ms for the code to compile and execute, 2024 kb of memory was used, and the runtime status was a success).

6. Implementation and Code:

App.js:

```
import "./App.css";
```

```
import Landing from "./components/Landing";

function App() {
  return <Landing />;
}

export default App;
```

index.js:

```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root")
);
```

Build the Code Editor Component: CodeEditorWindow.js

```
import React, { useState } from "react";

import Editor from "@monaco-editor/react";

const CodeEditorWindow = ({ onChange, language, code, theme }) => {
  const [value, setValue] = useState(code || "");

  const handleEditorChange = (value) => {
    setValue(value);
    onChange("code", value);
  };

  return (
    <div className="overlay rounded-md overflow-hidden text-lg w-full h-full shadow-4xl">
      <Editor
        options={{
          scrollBeyondLastLine:false,
          fontSize:"20px"
        }}
        height="85vh"
```

```
      width={`100%`}
      language={language || "cpp"}
      value={value}
      theme={theme}
      defaultValue="// some comment"
      onChange={handleEditorChange}
    />
   </div>
  );
};
export default CodeEditorWindow;
```

The Editor components comes from the @monaco-editor/react package.
The Editor component takes in a bunch of props:

- **language:** The language for which we require syntax highlighting and intellisense.
- **theme:** The colors and background of code snippet.
- **value:** The actual code value that goes into the code editor
- **onChange:** This is triggered when the value in the code editor changes. We need to save the changed value in the state so that we can, later on, call the Judge0 API to compile it.

The editor receives the props onChange, language, code, and theme from its parent component, which is Landing.js. Every time the value in the code editor changes, we call the onChange handler that is present in the parent Landing component.

Build the Landing Component: Landing.js

The landing component mostly consists of 3 parts:

- The Actions Bar which has the Languages and Themes drop-downs components.
- The Code Editor Window component
- The Output and Custom Input components

```
import React, { useEffect, useState } from "react";
import CodeEditorWindow from "./CodeEditorWindow";
import axios from "axios";
import { classnames } from "../utils/general";
import { languageOptions } from "../constants/languageOptions";

import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";

import { defineTheme } from "../lib/defineTheme";
import useKeyPress from "../hooks/useKeyPress";
import Footer from "./Footer";
import OutputWindow from "./OutputWindow";
import CustomInput from "./CustomInput";
import OutputDetails from "./OutputDetails";
import ThemeDropdown from "./ThemeDropdown";
import LanguagesDropdown from "./LanguagesDropdown";

const cppDefault = `// First Program
#include <iostream>
using namespace std;
int main() {
   cout << "Hello IWP!" << endl;
   return 0;
}`;

const Landing = () => {
  const [code, setCode] = useState(cppDefault);
  const [customInput, setCustomInput] = useState("");
  const [outputDetails, setOutputDetails] = useState(null);
  const [processing, setProcessing] = useState(null);
  const [theme, setTheme] = useState("brilliance-black");   // Default theme: brilliance-black
  const [language, setLanguage] = useState(languageOptions[7]);  // 7 for Default-cpp

  const enterPress = useKeyPress("Enter");
  const ctrlPress = useKeyPress("Control");

  const onSelectChange = (s1) => {
    console.log("selected Option...", s1);
    setLanguage(s1);
  };

  useEffect(() => {
    if (enterPress && ctrlPress) {
      console.log("enterPress", enterPress);
      console.log("ctrlPress", ctrlPress);
```

```
    handleCompile();
  }
}, [ctrlPress, enterPress]);
const onChange = (action, data) => {
  switch (action) {
    case "code": {
      setCode(data);
      break;
    }
    default: {
      console.warn("case not handled!", action, data);
    }
  }
};
const handleCompile = () => {
  setProcessing(true);
  const formData = {
    language_id: language.id,
    // encode source code in base64
    source_code: btoa(code),
    stdin: btoa(customInput),
  };

  const options = {
    method: "POST",
    url: 'https://judge0-ce.p.rapidapi.com/submissions',
    params: { base64_encoded: "true", fields: "*" },
    headers: {
      "content-type": "application/json",
      "Content-Type": "application/json",
      'X-RapidAPI-Host': 'judge0-ce.p.rapidapi.com',
      'X-RapidAPI-Key': '2ba277242amsh391b9714362c3d0p1e04b0jsndd276d18d17a',
    },
    data: formData,
  };

  axios
    .request(options)
    .then(function (response) {
      console.log("res.data", response.data);
      const token = response.data.token;
      checkStatus(token);
    })
    .catch((err) => {
      let error = err.response ? err.response.data : err;
      // get error status
```

```
      let status = err.response.status;
      console.log("status", status);
      if (status === 429) {
        console.log("Too many requests", status);

        showErrorToast(
          `Quota of 100 requests exceeded for the Day!`,
          10000
        );
      }
      setProcessing(false);
      console.log("catch block...", error);
    });
};

const checkStatus = async (token) => {
  const options = {
    method: "GET",
    url: 'https://judge0-ce.p.rapidapi.com/submissions' + "/" + token,
    params: { base64_encoded: "true", fields: "*" },
    headers: {
      'X-RapidAPI-Host': 'judge0-ce.p.rapidapi.com',
      'X-RapidAPI-Key': '2ba277242amsh391b9714362c3d0p1e04b0jsndd276d18d17a',
    },
  };
  try {
    let response = await axios.request(options);
    let statusId = response.data.status?.id;

    // Processed - we have a result
    if (statusId === 1 || statusId === 2) {
      // still processing
      setTimeout(() => {
        checkStatus(token);
      }, 2000);
      return;
    } else {
      setProcessing(false);
      setOutputDetails(response.data);
      showSuccessToast(`Compiled Successfully!`);
      console.log("response.data", response.data);
      return;
    }
  } catch (err) {
    console.log("err", err);
    setProcessing(false);
```

```
    showErrorToast();
  }
};

function handleThemeChange(th) {
  const theme = th;
  console.log("theme...", theme);

  if (["dark", "vs-dark"].includes(theme.value)) {
    setTheme(theme);
  } else {
    defineTheme(theme.value).then((_) => setTheme(theme));
  }
}

// Default-Theme: Brilliance Black
useEffect(() => {
  defineTheme("brilliance-black").then((_) =>
    setTheme({ value: "brilliance-black", label: "Select Theme" })
  );
}, []);

const showSuccessToast = (msg) => {
  toast.success(msg || `Compiled Successfully!`, {
    position: "top-right",
    autoClose: 1000,
    hideProgressBar: false,
    closeOnClick: true,
    pauseOnHover: true,
    draggable: true,
    progress: undefined,
  });
};
const showErrorToast = (msg, timer) => {
  toast.error(msg || `Something went wrong! Please try again.`, {
    position: "top-right",
    autoClose: timer ? timer : 1000,
    hideProgressBar: false,
    closeOnClick: true,
    pauseOnHover: true,
    draggable: true,
    progress: undefined,
  });
};

return (
```

```
    <>
      <ToastContainer
        position="top-right"
        autoClose={2000}
        hideProgressBar={false}
        newestOnTop={false}
        closeOnClick
        rtl={false}
        pauseOnFocusLoss
        draggable
        pauseOnHover
      />

      <div className="flex flex-row">
        <div className="px-4 py-2">
          <LanguagesDropdown onSelectChange={onSelectChange} />
        </div>
        <div className="px-4 py-2">
          <ThemeDropdown handleThemeChange={handleThemeChange} theme={theme} />
        </div>
      </div>
      <div className="flex flex-row text-lg space-x-4 items-start px-4 py-4">
        <div className="flex flex-col w-full text-lg h-full justify-start items-end">
          <CodeEditorWindow
            code={code}
            onChange={onChange}
            language={language?.value}
            theme={theme.value}
          />
        </div>

        <div className="right-container flex flex-shrink-0 w-[30%] flex-col">
          <OutputWindow outputDetails={outputDetails} />
          <div className="flex flex-col items-end">
            <CustomInput
              customInput={customInput}
              setCustomInput={setCustomInput}
            />
            <button
              onClick={handleCompile}
              disabled={!code}
              className={classnames(
                "mt-4 border-2 border-black z-10 rounded-md shadow-
[5px_5px_0px_0px_rgba(0,0,0)] px-4 py-2 hover:shadow transition duration-200 bg-white flex-
shrink-0",
                !code ? "opacity-50" : ""
```

```
            )}
          >
            {processing ? "Processing..." : "Compile and Execute"}
          </button>
        </div>
        {outputDetails && <OutputDetails outputDetails={outputDetails} />}
      </div>
    </div>
    <Footer />
  </>
 );
};
export default Landing;
```

The CodeEditorWindow component also takes into account the language prop, which is the current selected language we need syntax highlighting and intellisense for.
I've created a languageOptions array which keeps track of the accepted language props by the Monaco Editor and also handles the compilation (we keep track of the languageId that is accepted by judge0 APIs).

constants/languageOptions.js

```
export const languageOptions = [
  {
    id: 63,
    name: "JavaScript (Node.js 12.14.0)",
    label: "JavaScript (Node.js 12.14.0)",
    value: "javascript",
  },
  {
    id: 45,
    name: "Assembly (NASM 2.14.02)",
    label: "Assembly (NASM 2.14.02)",
    value: "assembly",
  },
  {
    id: 46,
    name: "Bash (5.0.0)",
    label: "Bash (5.0.0)",
    value: "bash",
  },
```

```
{
  id: 47,
  name: "Basic (FBC 1.07.1)",
  label: "Basic (FBC 1.07.1)",
  value: "basic",
},
{
  id: 75,
  name: "C (Clang 7.0.1)",
  label: "C (Clang 7.0.1)",
  value: "c",
},
{
  id: 76,
  name: "C++ (Clang 7.0.1)",
  label: "C++ (Clang 7.0.1)",
  value: "cpp",
},
{
  id: 48,
  name: "C (GCC 7.4.0)",
  label: "C (GCC 7.4.0)",
  value: "c",
},
{
  id: 52,
  name: "C++ (GCC 7.4.0)",
  label: "C++ (GCC 7.4.0)",
  value: "cpp",
},
{
  id: 49,
  name: "C (GCC 8.3.0)",
  label: "C (GCC 8.3.0)",
  value: "c",
},
{
  id: 53,
  name: "C++ (GCC 8.3.0)",
  label: "C++ (GCC 8.3.0)",
  value: "cpp",
},
{
  id: 50,
  name: "C (GCC 9.2.0)",
  label: "C (GCC 9.2.0)",
```

```
  value: "c",
},
{
  id: 54,
  name: "C++ (GCC 9.2.0)",
  label: "C++ (GCC 9.2.0)",
  value: "cpp",
},
{
  id: 86,
  name: "Clojure (1.10.1)",
  label: "Clojure (1.10.1)",
  value: "clojure",
},
{
  id: 51,
  name: "C# (Mono 6.6.0.161)",
  label: "C# (Mono 6.6.0.161)",
  value: "csharp",
},
{
  id: 77,
  name: "COBOL (GnuCOBOL 2.2)",
  label: "COBOL (GnuCOBOL 2.2)",
  value: "cobol",
},
{
  id: 55,
  name: "Common Lisp (SBCL 2.0.0)",
  label: "Common Lisp (SBCL 2.0.0)",
  value: "lisp",
},
{
  id: 56,
  name: "D (DMD 2.089.1)",
  label: "D (DMD 2.089.1)",
  value: "d",
},
{
  id: 57,
  name: "Elixir (1.9.4)",
  label: "Elixir (1.9.4)",
  value: "elixir",
},
{
  id: 58,
```

```
  name: "Erlang (OTP 22.2)",
  label: "Erlang (OTP 22.2)",
  value: "erlang",
},
{
  id: 44,
  label: "Executable",
  name: "Executable",
  value: "exe",
},
{
  id: 87,
  name: "F# (.NET Core SDK 3.1.202)",
  label: "F# (.NET Core SDK 3.1.202)",
  value: "fsharp",
},
{
  id: 59,
  name: "Fortran (GFortran 9.2.0)",
  label: "Fortran (GFortran 9.2.0)",
  value: "fortran",
},
{
  id: 60,
  name: "Go (1.13.5)",
  label: "Go (1.13.5)",
  value: "go",
},
{
  id: 88,
  name: "Groovy (3.0.3)",
  label: "Groovy (3.0.3)",
  value: "groovy",
},
{
  id: 61,
  name: "Haskell (GHC 8.8.1)",
  label: "Haskell (GHC 8.8.1)",
  value: "haskell",
},
{
  id: 62,
  name: "Java (OpenJDK 13.0.1)",
  label: "Java (OpenJDK 13.0.1)",
  value: "java",
},
```

```
{
  id: 78,
  name: "Kotlin (1.3.70)",
  label: "Kotlin (1.3.70)",
  value: "kotlin",
},
{
  id: 64,
  name: "Lua (5.3.5)",
  label: "Lua (5.3.5)",
  value: "lua",
},

{
  id: 79,
  name: "Objective-C (Clang 7.0.1)",
  label: "Objective-C (Clang 7.0.1)",
  value: "objectivec",
},
{
  id: 65,
  name: "OCaml (4.09.0)",
  label: "OCaml (4.09.0)",
  value: "ocaml",
},
{
  id: 66,
  name: "Octave (5.1.0)",
  label: "Octave (5.1.0)",
  value: "octave",
},
{
  id: 67,
  name: "Pascal (FPC 3.0.4)",
  label: "Pascal (FPC 3.0.4)",
  value: "pascal",
},
{
  id: 85,
  name: "Perl (5.28.1)",
  label: "Perl (5.28.1)",
  value: "perl",
},
{
  id: 68,
```

```
  name: "PHP (7.4.1)",
  label: "PHP (7.4.1)",
  value: "php",
},
{
  id: 43,
  label: "Plain Text",
  name: "Plain Text",
  value: "text",
},
{
  id: 69,
  name: "Prolog (GNU Prolog 1.4.5)",
  label: "Prolog (GNU Prolog 1.4.5)",
  value: "prolog",
},
{
  id: 70,
  name: "Python (2.7.17)",
  label: "Python (2.7.17)",
  value: "python",
},
{
  id: 71,
  name: "Python (3.8.1)",
  label: "Python (3.8.1)",
  value: "python",
},
{
  id: 80,
  name: "R (4.0.0)",
  label: "R (4.0.0)",
  value: "r",
},
{
  id: 72,
  name: "Ruby (2.7.0)",
  label: "Ruby (2.7.0)",
  value: "ruby",
},
{
  id: 73,
  name: "Rust (1.40.0)",
  label: "Rust (1.40.0)",
  value: "rust",
},
```

```
  {
    id: 81,
    name: "Scala (2.13.2)",
    label: "Scala (2.13.2)",
    value: "scala",
  },
  {
    id: 82,
    name: "SQL (SQLite 3.27.2)",
    label: "SQL (SQLite 3.27.2)",
    value: "sql",
  },
  {
    id: 83,
    name: "Swift (5.2.3)",
    label: "Swift (5.2.3)",
    value: "swift",
  },
  {
    id: 74,
    name: "TypeScript (3.7.4)",
    label: "TypeScript (3.7.4)",
    value: "typescript",
  },
  {
    id: 84,
    name: "Visual Basic.Net (vbnc 0.0.0.5943)",
    label: "Visual Basic.Net (vbnc 0.0.0.5943)",
    value: "vbnet",
  },
];
```

Every languageOptions object contains an id, name, label and value keys. The entire languageOptions array can be taken and put inside the dropdown and supplied as options.
Whenever the state of the dropdown changes, the onSelectChange method keeps track of the selected id and changes the state appropriately.

<u>LanguageDropdown component:</u> LanguageDropdown.js

```
import React from "react";
import Select from "react-select";
```

```
import { customStyles } from "../constants/customStyles";
import { languageOptions } from "../constants/languageOptions";

const LanguagesDropdown = ({ onSelectChange }) => {
  return (
    <Select
      placeholder={`Select Language`}
      options={languageOptions}
      styles={customStyles}
      //defaultValue={languageOptions[7]}  // for Default-Laguage: cpp
      onChange={(selectedOption) => onSelectChange(selectedOption)}
    />
  );
};

export default LanguagesDropdown;
```

For the dropdown, we are going to use the package react-select that takes care of the dropdowns and their change handlers.
React select takes defaultValue and options as the major parameters. options is an array (and we are going to pass languageOptions here) that automatically shows all the relevant dropdown values.
The defaultValue prop is the default value which is provided to the component. We are going to keep C++ as the default language (which is the seventh position in our array of languages).
Whenever the user changes the language, we change the language with the onSelectChange callback:

ThemeDropdown Component: ThemeDropdown.js

The ThemeDropdown component is actually very similar to the LanguageDropdown component (with the UI and react-select package).

```
import React from "react";
import Select from "react-select";
import monacoThemes from "monaco-themes/themes/themelist";
import { customStyles } from "../constants/customStyles";

const ThemeDropdown = ({ handleThemeChange, theme }) => {
```

```
  return (
    <Select
      placeholder={`Select Theme`}
      // options={languageOptions}
      options={Object.entries(monacoThemes).map(([themeId, themeName]) => ({
        label: themeName,
        value: themeId,
        key: themeId,
      }))}
      value={theme}
      styles={customStyles}
      onChange={handleThemeChange}
    />
  );
};

export default ThemeDropdown;
```

Here, we are going to use the package monacoThemes to help us with the different beautiful themes available on the internet for Monaco Editor.

lib/defineTheme.js:

```
import { loader } from "@monaco-editor/react";

const monacoThemes = {
  active4d: "Active4D",
  "all-hallows-eve": "All Hallows Eve",
  amy: "Amy",
  "birds-of-paradise": "Birds of Paradise",
  blackboard: "Blackboard",
  "brilliance-black": "Brilliance Black",
  "brilliance-dull": "Brilliance Dull",
  "chrome-devtools": "Chrome DevTools",
  "clouds-midnight": "Clouds Midnight",
  clouds: "Clouds",
  cobalt: "Cobalt",
  dawn: "Dawn",
  dreamweaver: "Dreamweaver",
  eiffel: "Eiffel",
  "espresso-libre": "Espresso Libre",
  github: "GitHub",
  idle: "IDLE",
```

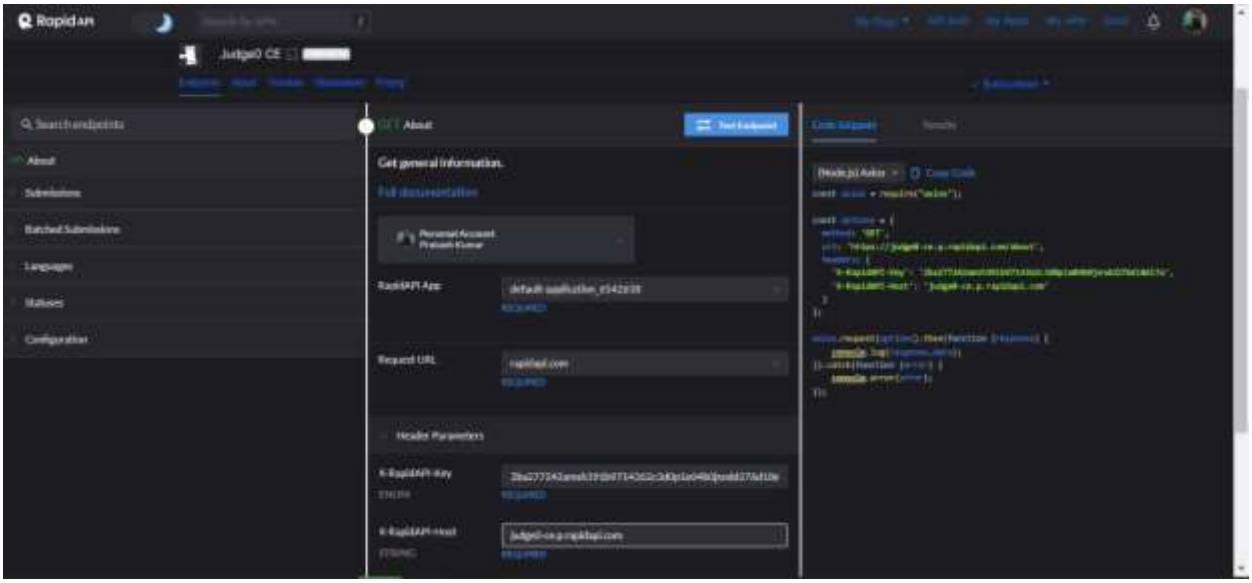
```
  katzenmilch: "Katzenmilch",
  "kuroir-theme": "Kuroir Theme",
  lazy: "LAZY",
  "magicwb--amiga-": "MagicWB (Amiga)",
  "merbivore-soft": "Merbivore Soft",
  merbivore: "Merbivore",
  "monokai-bright": "Monokai Bright",
  monokai: "Monokai",
  "night-owl": "Night Owl",
  "oceanic-next": "Oceanic Next",
  "pastels-on-dark": "Pastels on Dark",
  "slush-and-poppies": "Slush and Poppies",
  "solarized-dark": "Solarized-dark",
  "solarized-light": "Solarized-light",
  spacecadet: "SpaceCadet",
  sunburst: "Sunburst",
  "textmate--mac-classic-": "Textmate (Mac Classic)",
  "tomorrow-night-blue": "Tomorrow-Night-Blue",
  "tomorrow-night-bright": "Tomorrow-Night-Bright",
  "tomorrow-night-eighties": "Tomorrow-Night-Eighties",
  "tomorrow-night": "Tomorrow-Night",
  tomorrow: "Tomorrow",
  twilight: "Twilight",
  "upstream-sunburst": "Upstream Sunburst",
  "vibrant-ink": "Vibrant Ink",
  "xcode-default": "Xcode_default",
  zenburnesque: "Zenburnesque",
  iplastic: "iPlastic",
  idlefingers: "idleFingers",
  krtheme: "krTheme",
  monoindustrial: "monoindustrial",
};

const defineTheme = (theme) => {
  return new Promise((res) => {
    Promise.all([
      loader.init(),
      import(`monaco-themes/themes/${monacoThemes[theme]}.json`),
    ]).then(([monaco, themeData]) => {
      monaco.editor.defineTheme(theme, themeData);
      res();
    });
  });
};

export { defineTheme };
```

Compile the Code with Judge0:

For compiling our code, we are going to use Judge0. Judge0 is a simple, open-source code execution system that we can interact with.

We can make a simple API call with some parameters (source code, language ID) and get the output as a response.

The dashboard looks something like this:



REACT_APP_RAPID_API_HOST = YOUR_HOST_URL
REACT_APP_RAPID_API_KEY = YOUR_SECRET_KEY
REACT_APP_RAPID_API_URL = YOUR_SUBMISSIONS_URL

The SUBMISSIONS_URL is the URL that we are going to use. It basically consists of your host , followed by the /submission route.

https://judge0-ce.p.rapidapi.com/submissions will be the submissions URL in our case.

Compilation Flow and Logic:

The flow of compiling is as follows:

- When the button Compile and Execute is clicked, a method handleCompile() is called.
- The handleCompile() function calls our Judge0 RapidAPI backend on the submissions URL with languageId , source_code, and stdin (customInput in our case) as the body params.
- the options also takes in the host and the secret as headers.
- base64_encoded and fields are optional parameters that can be passed.
- The submission POST request registers our request in the server and creates a process. The response of the post request is a token that can be further used to check the status of our execution. (There are various statuses – Processing, Accepted, Time Limit Exceeded, Runtime Exceptions and more.)
- Once our results are returned, we can conditionally check if the results are success or failure and show the results on our output screens.

When there's a successful response and we have a token, we call the checkStatus() method to poll the /submissions/${token} route.

statuses.js:

```
export const statuses = [
  {
    id: 1,
    description: "In Queue",
  },
  {
    id: 2,
    description: "Processing",
  },
  {
    id: 3,
```

```
  description: "Accepted",
},
{
  id: 4,
  description: "Wrong Answer",
},
{
  id: 5,
  description: "Time Limit Exceeded",
},
{
  id: 6,
  description: "Compilation Error",
},
{
  id: 7,
  description: "Runtime Error (SIGSEGV)",
},
{
  id: 8,
  description: "Runtime Error (SIGXFSZ)",
},
{
  id: 9,
  description: "Runtime Error (SIGFPE)",
},
{
  id: 10,
  description: "Runtime Error (SIGABRT)",
},
{
  id: 11,
  description: "Runtime Error (NZEC)",
},
{
  id: 12,
  description: "Runtime Error (Other)",
},
{
  id: 13,
  description: "Internal Error",
},
{
  id: 14,
  description: "Exec Format Error",
},
```

```
];
```

CustomInput.js:

```
import React from "react";
import { classnames } from "../utils/general";

const CustomInput = ({ customInput, setCustomInput }) => {
  return (
    <>
      {" "}
      <textarea
        rows="5"
        value={customInput}
        onChange={(e) => setCustomInput(e.target.value)}
        placeholder={`Custom input`}
        className={classnames(
          "focus:outline-none w-full border-2 border-white text-white text-lg z-10 rounded-md shadow-[5px_5px_0px_0px_rgba(0,0,0)] px-4 py-2 hover:shadow transition duration-200 bg-[#0d0d0d] mt-2"
        )}
      ></textarea>
    </>
  );
};

export default CustomInput;
```

OutputWindow.js:

```
import React from "react";

const OutputWindow = ({ outputDetails }) => {
  const getOutput = () => {
    let statusId = outputDetails?.status?.id;

    if (statusId === 6) {
      // compilation error
      return (
        // <pre className="px-2 py-1 font-normal text-xs text-red-500">
        <pre className="px-2 py-1 font-normal text-lg text-red-500">
```

```
        {atob(outputDetails?.compile_output)}
      </pre>
    );
  } else if (statusId === 3) {
    return (
      <pre className="px-2 py-1 font-normal text-lg text-green-500">
        {atob(outputDetails.stdout) !== null
          ? `${atob(outputDetails.stdout)}`
          : null}
      </pre>
    );
  } else if (statusId === 5) {
    return (
      <pre className="px-2 py-1 font-normal text-lg text-red-500">
        {`Time Limit Exceeded`}
      </pre>
    );
  } else {
    return (
      <pre className="px-2 py-1 font-normal text-lg text-red-500">
        {atob(outputDetails?.stderr)}
      </pre>
    );
  }
 };
 return (
  <>
   {/* <h1 className="font-bold text-xl bg-clip-text text-transparent bg-gradient-to-r from-slate-900 to-slate-700 mb-2"> */}
   <h1 className="font-bold text-2xl bg-clip-text text-transparent bg-gradient-to-r from-slate-900 to-slate-700 mb-2">
     Output
   </h1>
   <div className="w-full h-56 bg-[#0d0d0d] rounded-md text-white font-normal text-sm overflow-y-auto">
     {outputDetails ? <>{getOutput()}</> : null}
   </div>
  </>
 );
};

export default OutputWindow;
```

OutputDetails.js:

```
import React from "react";

const OutputDetails = ({ outputDetails }) => {
  return (
    <div className="metrics-container mt-4 flex flex-col space-y-3">
      <p className="text-lg">
        Status:{" "}
        <span className="font-semibold text-lg px-2 py-1 rounded-md bg-gray-100">
          {outputDetails?.status?.description}
        </span>
      </p>
      <p className="text-lg">
        Memory:{" "}
        <span className="font-semibold text-lg px-2 py-1 rounded-md bg-gray-100">
          {outputDetails?.memory}
        </span>
      </p>
      <p className="text-lg">
        Time:{" "}
        <span className="font-semibold text-lg px-2 py-1 rounded-md bg-gray-100">
          {outputDetails?.time}
        </span>
      </p>
    </div>
  );
};

export default OutputDetails;
```

The time, memory and status.description are all received from the API response which are then stored in outputDetails and displayed.

hooks/useKeyPress.js:

The last thing in the application is using ctrl+enter to compile our code. For this, we are going to create a custom hook to listen for various keyboard events on our web application.

```
import React, { useState } from "react";
```

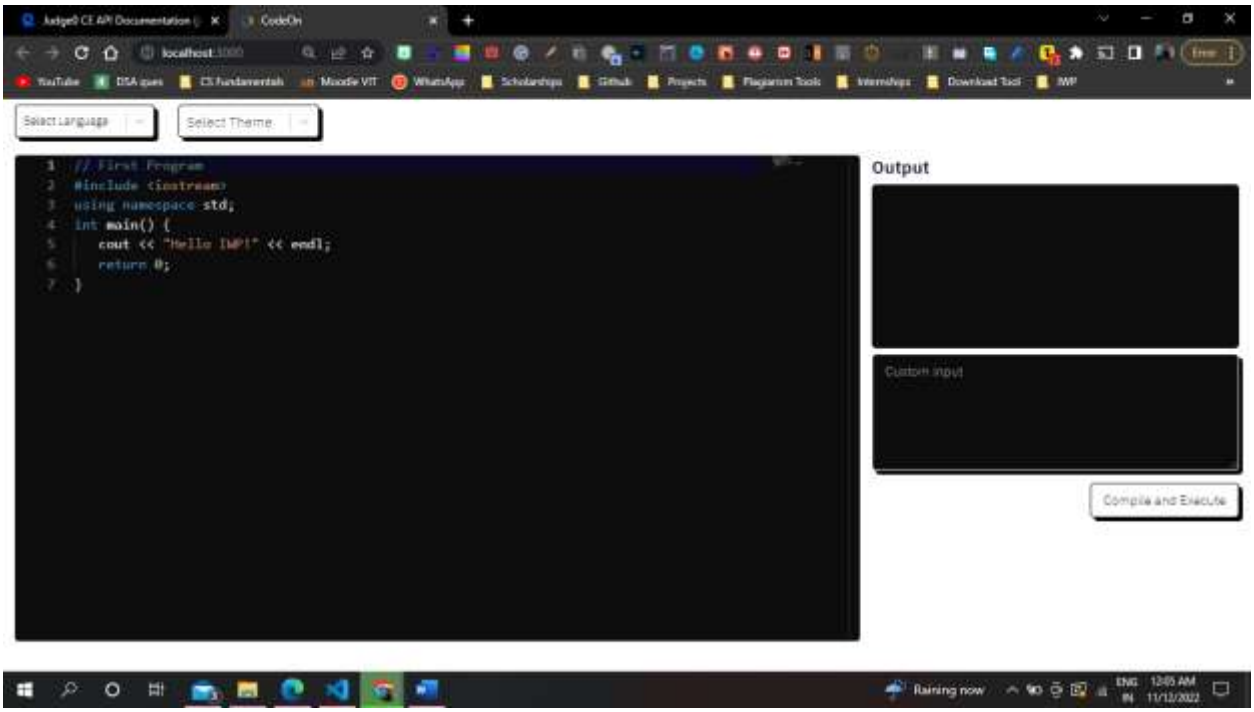
```
const useKeyPress = function (targetKey) {
  const [keyPressed, setKeyPressed] = useState(false);

  function downHandler({ key }) {
    if (key === targetKey) {
      setKeyPressed(true);
    }
  }

  const upHandler = ({ key }) => {
    if (key === targetKey) {
      setKeyPressed(false);
    }
  };

  React.useEffect(() => {
    document.addEventListener("keydown", downHandler);
    document.addEventListener("keyup", upHandler);

    return () => {
      document.removeEventListener("keydown", downHandler);
      document.removeEventListener("keyup", upHandler);
    };
  });

  return keyPressed;
};

export default useKeyPress;
```

customStyles.js:

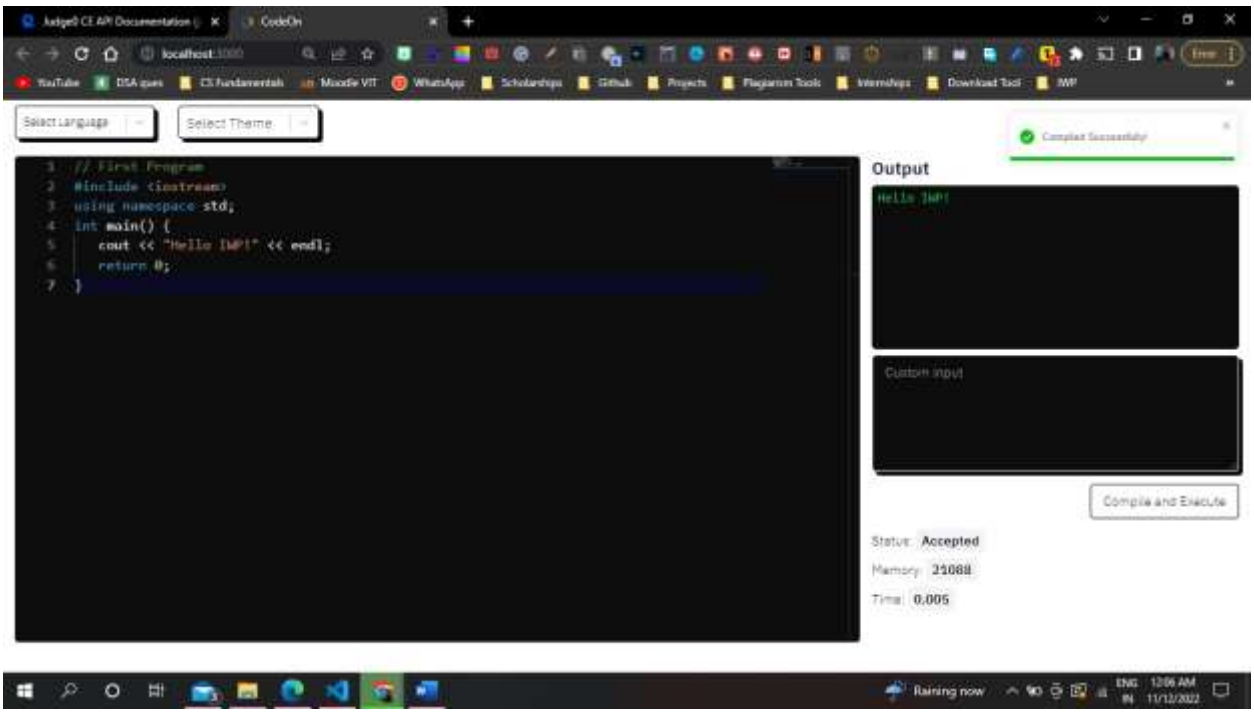
```
export const customStyles = {
  control: (styles) => ({
    ...styles,
    width: "100%",
    maxWidth: "14rem",
    minWidth: "12rem",
    borderRadius: "5px",
    color: "#000",
    fontSize: "1.2rem",
    lineHeight: "1.75rem",
    backgroundColor: "#FFFFFF",
```

```
    cursor: "pointer",
    border: "2px solid #000000",
    boxShadow: "5px 5px 0px 0px rgba(0,0,0);",
    ":hover": {
      border: "2px solid #000000",
      boxShadow: "none",
    },
  }),
  option: (styles) => {
    return {
      ...styles,
      color: "#000",
      fontSize: "1.0rem",
      lineHeight: "1.75rem",
      width: "100%",
      background: "#fff",
      ":hover": {
        backgroundColor: "rgb(243 244 246)",
        color: "#000",
        cursor: "pointer",
      },
    };
  },
  menu: (styles) => {
    return {
      ...styles,
      backgroundColor: "#fff",
      maxWidth: "14rem",
      border: "2px solid #000000",
      borderRadius: "5px",
      boxShadow: "5px 5px 0px 0px rgba(0,0,0);",
    };
  },

  placeholder: (defaultStyles) => {
    return {
      ...defaultStyles,
      color: "#000",
      fontSize: "1.0rem",
      lineHeight: "1.75rem",
    };
  },
};
```

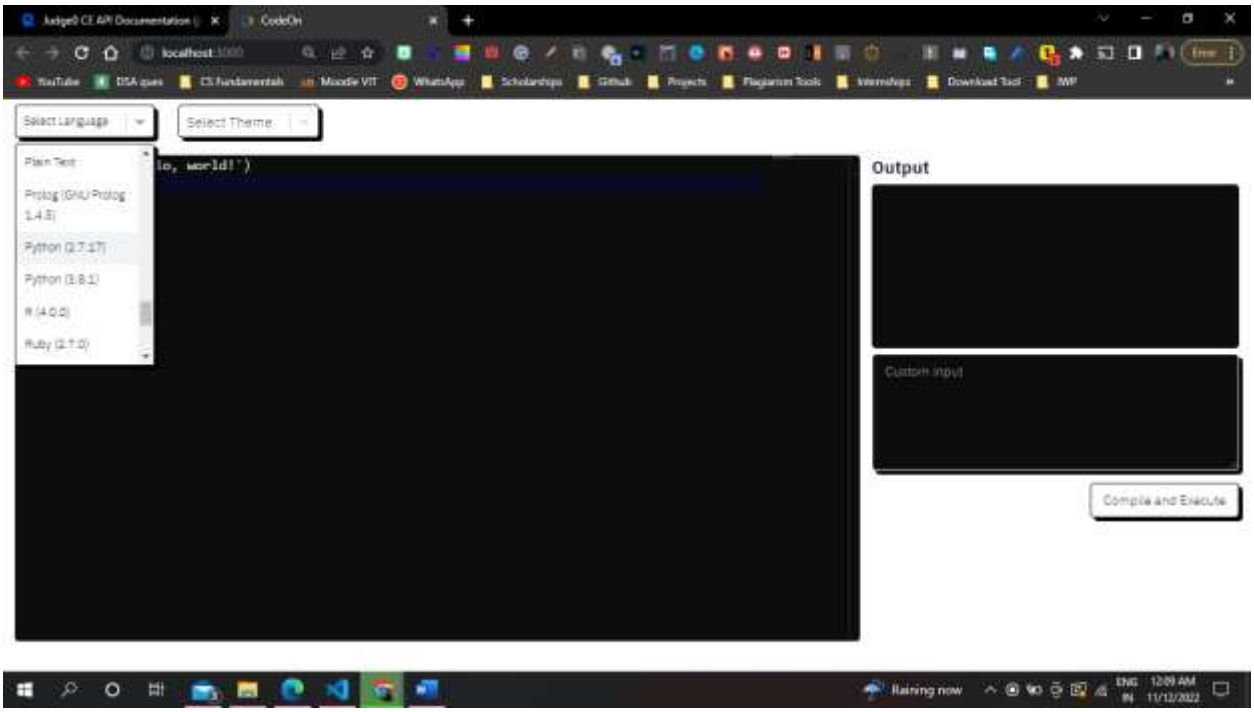
7. Screenshots of the Web Portal:



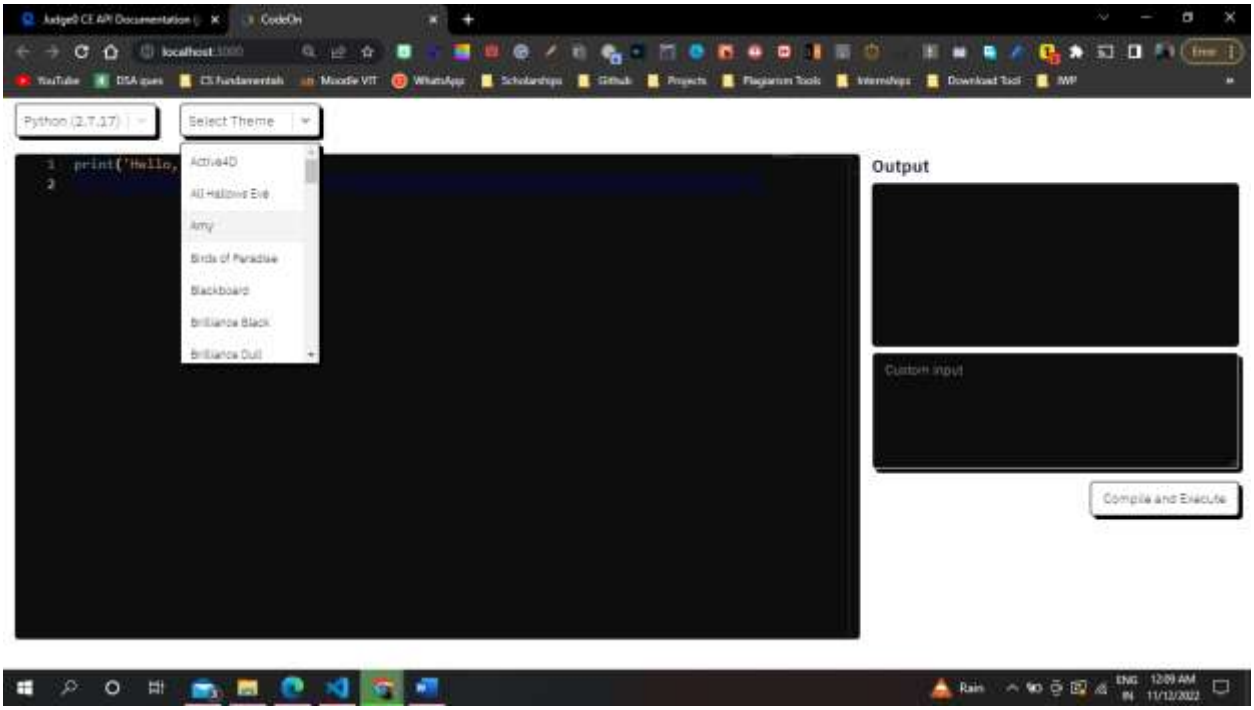
Compile and Execute:



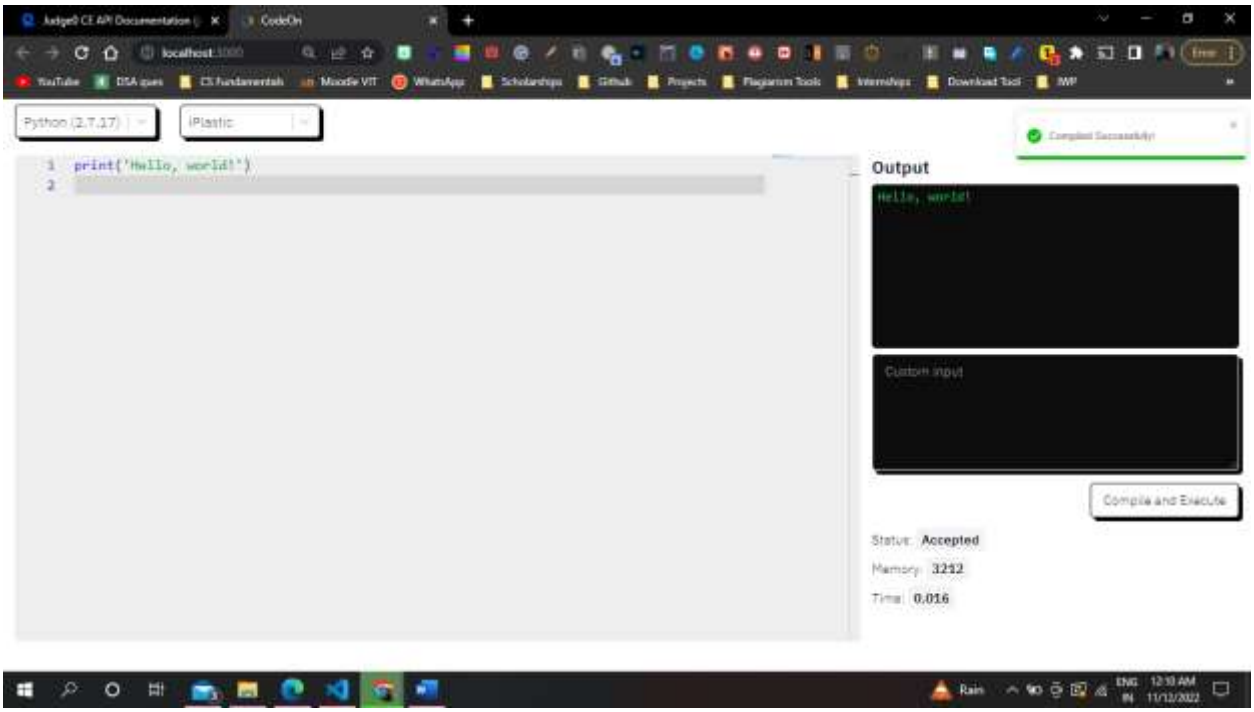
Select Language:



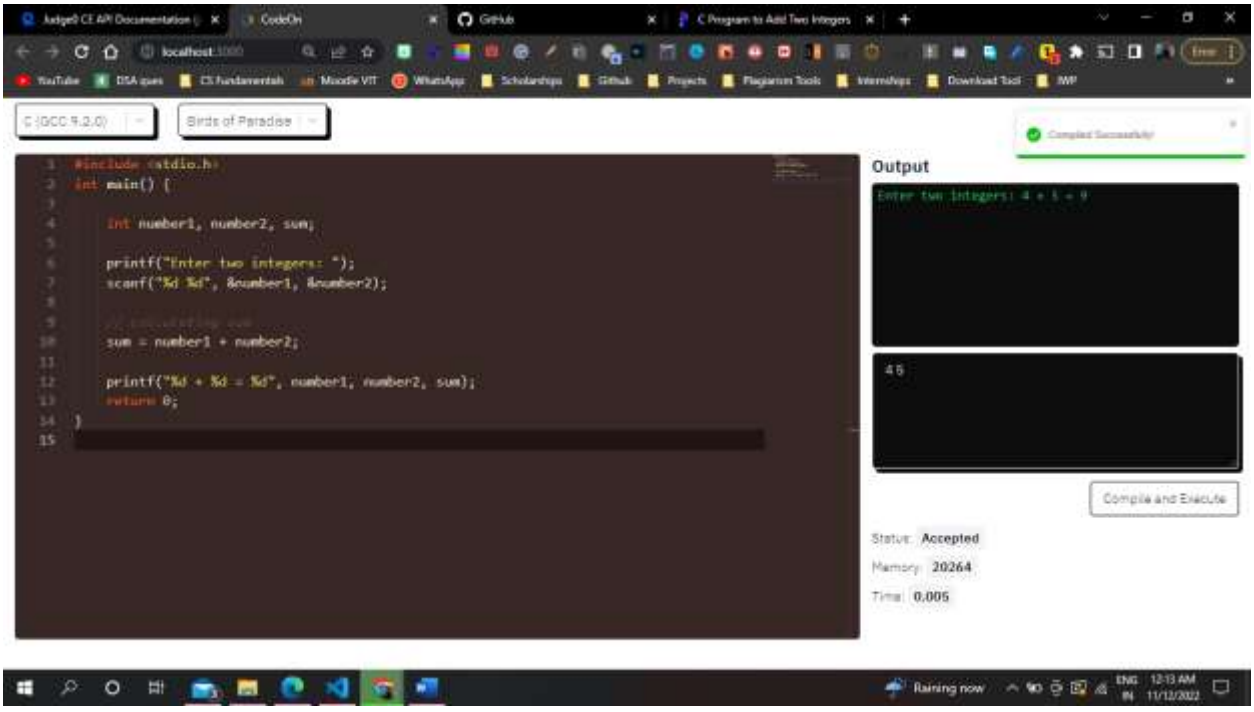
Select Theme:



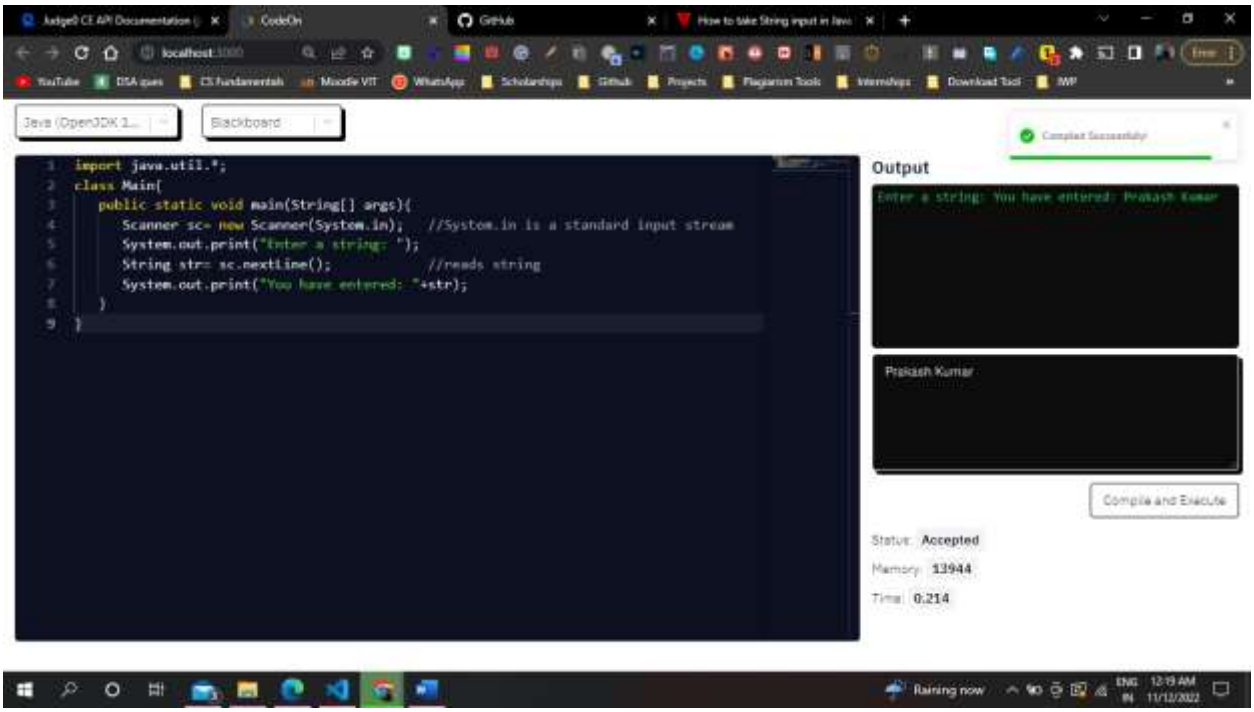
Compile and Execute:



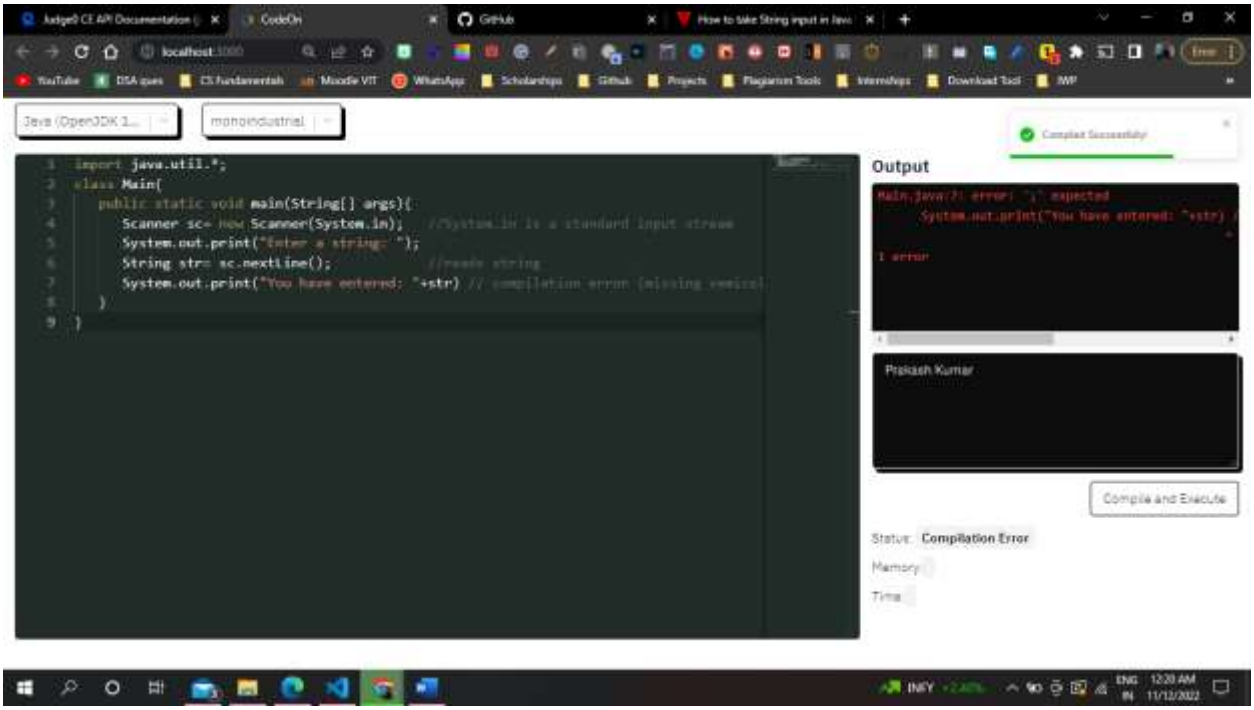
User Input: Integer user input



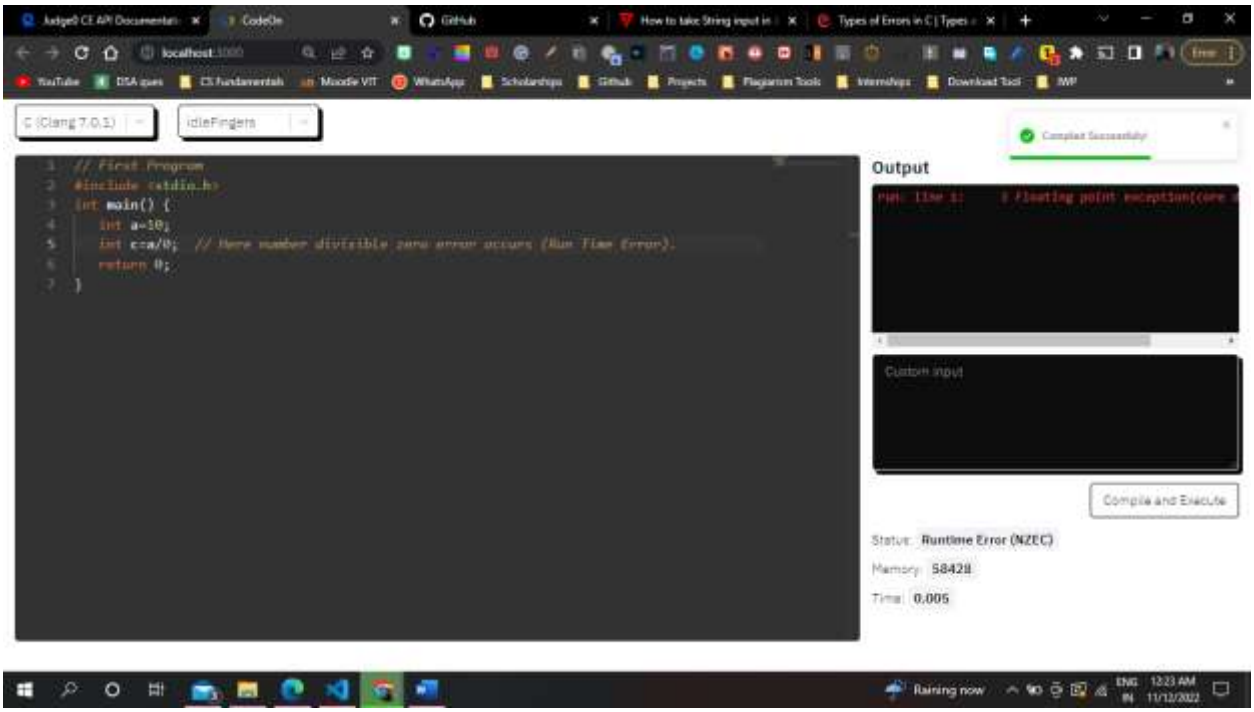
User Input: String user input



Error: Syntax Error



Error: Run Time Error



8. Conclusion:

In the end, we have:

- A code editor that can compile in 40+ languages.
- A theme switcher to change the look of our code editor.
- Interacting and Hosting APIs on RapidAPI.
- Using keyboard events in React using custom hooks.

9. References:

- React.js
- TailwindCSS

- Judge0
- RapidAPI
- Monaco Editor