Web Development Project
Task 2 Online Code Editor

Creating an online code editor running in browsers offers complete IDE functionalities, and it's a valuable skill-building project. It can potentially lead to a startup as a free interviewing platform. This project involves two main components: Backend API: This server-side component takes code and language as input and provides the output after executing the code on the server. Frontend Editor: On the frontend, you can choose a language, edit and modify the code. After editing, you make a post request to the backend API and display the output on the website.

Skills Required – HTML, CSS, ReactJS, Hosting Services

Creating an online code editor involves building a backend API to execute code and a frontend editor to interact with users. We will use React.js for the frontend and Node.js with Express for the backend. Additionally, we will deploy our application using a hosting service like Heroku.

Backend API (Node.js with Express)

1. **Initialize Node.js Project:** Create a new directory for project and run **npm init -y** to initialize a new Node.js project.
2. **Install Dependencies:** Install necessary packages such as Express, body-parser, and cors using npm:

npm install express body-parser cors

3. **Create server.js File:**

Create a **server.js** file to set up Express server and handle code execution requests.

```
// server.js

const express = require('express');

const bodyParser = require('body-parser');

const cors = require('cors');

const { exec } = require('child_process')

const app = express();

app.use(bodyParser.json());

app.use(cors());

app.post('/runcode', (req, res) => {

  const { code, language } = req.body;

  // Define language-specific commands for execution

  const commands = {

    javascript: 'node',

    python: 'python3',

    java: 'java',
```

```
    // Add more languages as needed
  };
  if (!commands[language]) {
    return res.status(400).send('Unsupported language');
  }
  exec(`${commands[language]} -e "${code}"`, (error, stdout, stderr) => {
    if (error) {
      return res.status(400).send(stderr);
    }
    res.send(stdout);
  });
});
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Frontend Editor (React.js)

1. Create React App: Initialize a new React app using Create React App.

```
npx create-react-app code-editor-app
```

2. Install Axios: Axios will be used to make HTTP requests to the backend API.

```
npm install axios
```

3. Modify src/App.js to include the code editor and execute code functionality:

```
import React, { useState } from 'react';
import axios from 'axios';
function App() {
  const [code, setCode] = useState('');
  const [language, setLanguage] = useState('javascript');
  const [output, setOutput] = useState('');
  const executeCode = async () => {
    try {
      const response = await axios.post('http://localhost:5000/runcode', {
```

```
      code,
      language,
    });
    setOutput(response.data);
  } catch (error) {
    console.error(error);
    setOutput('Error executing code');
  }
};
return (
  <div className="App">
    <h1>Online Code Editor</h1>
    <div>
      <label htmlFor="language">Select Language:</label>
      <select id="language" onChange={(e) => setLanguage(e.target.value)}>
        <option value="javascript">JavaScript</option>
        <option value="python">Python</option>
        <option value="java">Java</option>
        {/* Add more language options as needed */}
      </select>
    </div>
    <div>
      <label htmlFor="code">Enter Code:</label>
      <textarea id="code" value={code} onChange={(e) => setCode(e.target.value)}></textarea>
    </div>
    <button onClick={executeCode}>Execute Code</button>
    <div>
      <h3>Output:</h3>
      <pre>{output}</pre>
    </div>
  </div>
);
```

}

export default App;

Deploying to Heroku

1. Create a Procfile: Create a file named **Procfile** (without any file extension) in the root directory of project with the following content:

web: node server.js

2. Set Up Heroku: Install the Heroku CLI and create a new Heroku app.

heroku login

heroku create your-app-name

3. Deploy Your App: Push code to Heroku to deploy the app.

git push heroku main

This setup will allow users to write code in various languages, execute it through the backend API, and view the output directly on the website. Customize the UI and add additional features as needed for specific requirements.