**Scenario 1: Node Backend Based Sample Application and Debugging Using VS Code**

**Steps to Set Up a Node Backend Application:**

1. **Initialize a Node.js Project**:

mkdir node-backend-app

cd node-backend-app

npm init -y

1. **Install Express**:

npm install express

1. **Create a Simple Express Server**:

Create a file index.js:

const express = require('express');

const app = express();

const port = 3000;

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

res.send('Hello World!');

});

app.listen(port, () => {

console.log(`Server running on port ${port}`);

});

1. **Configure VS Code for Debugging**:
   * Open the project in VS Code.
   * Create a .vscode/launch.json file:

{

"version": "0.2.0",

"configurations": [

{

"type": "node",

"request": "launch",

"name": "Launch Node App",

"program": "${workspaceFolder}/index.js",

"outFiles": ["${workspaceFolder}/\*\*/\*.js"],

"console": "integratedTerminal"

}

]

}

1. **Run and Debug the Application**:
   * Press F5 in VS Code to start debugging the Node.js application.
   * Set breakpoints in the code, and the application will pause execution at those points, allowing you to inspect variables, evaluate expressions, etc.

**Scenario 2: React Typescript Based Application Running at Port 3000 and Node + Express Running at Port 8000**

**Steps to Set Up a React TypeScript Application:**

1. **Create a React TypeScript Project**:

npx create-react-app react-ts-app --template typescript

cd react-ts-app

1. **Run the React App**:
   * Start the React app on port 3000:

npm start

1. The app should now be running at http://localhost:3000.

**Steps to Set Up a Node + Express Backend:**

1. **Initialize the Node + Express Backend**:
   * In the same root folder (react-ts-app), create a backend folder:

mkdir backend

cd backend

npm init -y

1. **Install Express**:

npm install express

1. **Create an Express Server (running on port 8000)**:
   * Create a file backend/server.js:

const express = require('express');

const app = express();

const port = 8000;

app.get('/api', (req, res) => {

res.send({ message: 'Hello from Node backend!' });

});

app.listen(port, () => {

console.log(`Backend running on port ${port}`);

});

1. **Configure CORS in Express (if needed)**:
   * To allow communication between the React front-end and Node back-end, you might need to install cors:

npm install cors

* + Update server.js to use CORS:

const cors = require('cors');

app.use(cors({ origin: 'http://localhost:3000' }));

1. **Run the Node Server**:

node server.js

**Connect React App with Node Backend:**

1. **Create a Proxy in React App**:
   * In the package.json of the React app, add the following to enable proxying requests to the Node server:

"proxy": "http://localhost:8000",

1. **Make API Calls in React**:
   * Modify App.tsx to fetch data from the backend:

import React, { useEffect, useState } from 'react';

const App: React.FC = () => {

const [message, setMessage] = useState('');

useEffect(() => {

fetch('/api')

.then((response) => response.json())

.then((data) => setMessage(data.message));

}, []);

return <div>{message}</div>;

};

export default App;

**Scenario 3: WebdriverIO Based Test Script Debugging for Scenario 2**

**Steps to Set Up WebdriverIO:**

1. **Initialize WebdriverIO**:

npx wdio config

* + Choose Mocha or Jasmine as the test framework.
  + Choose the appropriate browser (e.g., Chrome).
  + The WebdriverIO configuration file (wdio.conf.js) will be generated.

1. **Install Required Packages**:

npm install @wdio/cli @wdio/local-runner @wdio/mocha-framework @wdio/spec-reporter @wdio/sync @wdio/chromedriver-service

1. **Write a WebdriverIO Test**:
   * In the test/specs folder, create a test script test.spec.js:

describe('React App', () => {

it('should load the page and display the message from backend', () => {

browser.url('http://localhost:3000');

const message = $('div').getText();

expect(message).toBe('Hello from Node backend!');

});

});

**Debugging WebdriverIO in VS Code:**

1. **Configure Debugging in VS Code**:
   * In the root folder, create a .vscode/launch.json:

{

"version": "0.2.0",

"configurations": [

{

"type": "node",

"request": "launch",

"name": "Debug WebdriverIO Test",

"program": "${workspaceFolder}/node\_modules/.bin/wdio",

"args": ["${workspaceFolder}/wdio.conf.js"],

"console": "integratedTerminal",

"internalConsoleOptions": "openOnSessionStart"

}

]

}

1. **Run the Debugger**:
   * Press F5 to start debugging the WebdriverIO tests.
   * Set breakpoints in the test code, and VS Code will pause execution, allowing you to inspect the state.

**Scenario 4: Connecting React, Node, and WebdriverIO**

This scenario is a combination of **Scenario 2** and **Scenario 3**:

* Your React front-end runs at http://localhost:3000, and the Node back-end runs at http://localhost:8000.
* WebdriverIO will interact with the front-end to verify that the backend integration works properly.

**Summary:**

1. **React TypeScript app** (port 3000) fetches data from the **Node backend** (port 8000).
2. **WebdriverIO tests** run on the front-end (React), checking the data received from the backend.