Lab 1: Guide to Using JSX and ES6

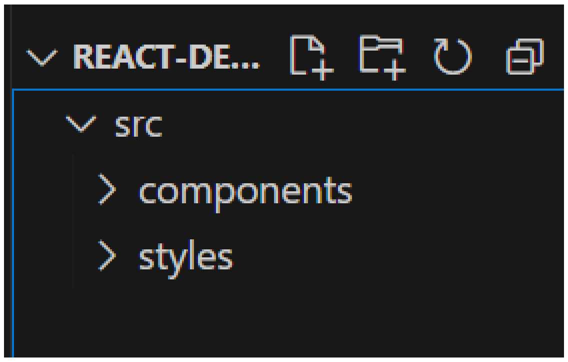
1. Building a React Project Using Webpack and Babel

Before you begin, you need to install npm and Node.js on your machine.

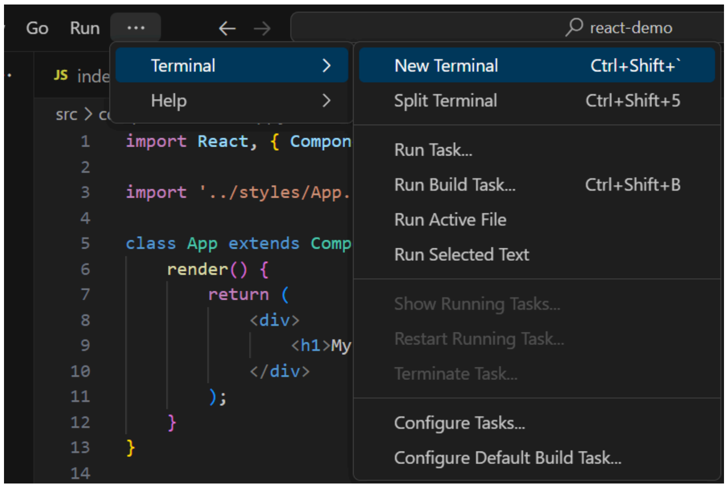
1.1. Directory Structure

Create a project directory with a name of your choice.

For example, let's create a project called **react-demo**. Inside the project directory, create another directory called src, and within that directory, create two subdirectories named components and styles. The directory structure will be as follows:



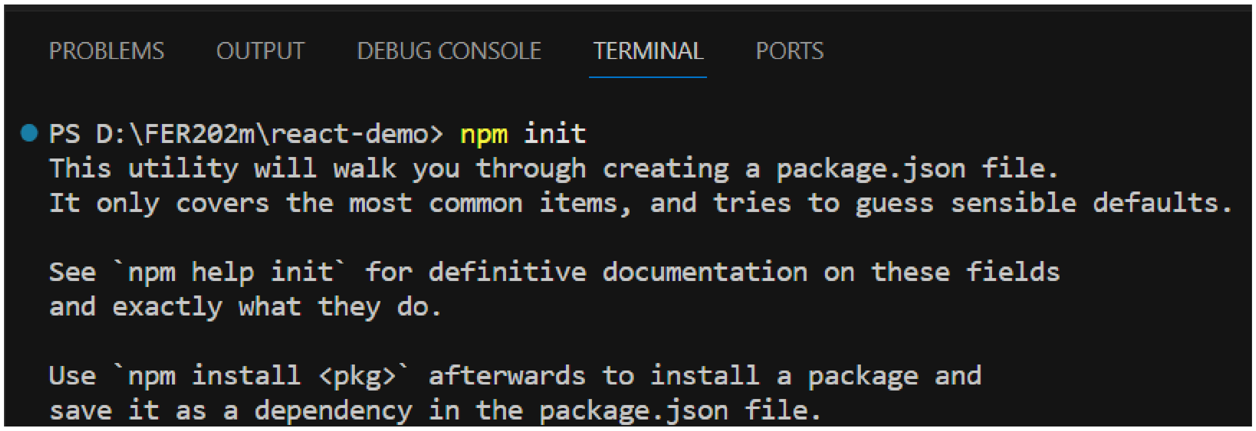
1.2. Initializing the Project



```
import React, { Compon

import '../styles/App.

class App extends Comp
    render() {
        return (
            <div>
                <h1>My
            </div>
        );
    }
}
```

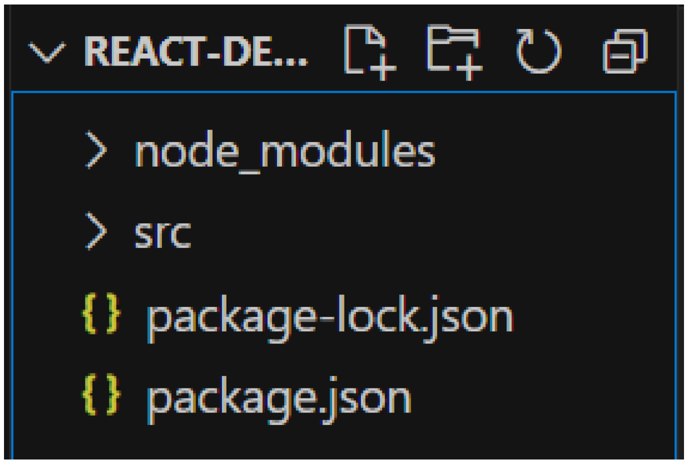
All projects using a package manager (npm) need to be initialized. To initialize the project, use the following command:

```
npm init
```



```
PS D:\FER202m\react-demo> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.
```

After executing this command, a file named **package.json** will be created in the react-demo project directory.



```
REACT-DE...
  > node_modules
  > src
  {} package-lock.json
  {} package.json
```

When running npm init, you will be prompted with some questions related to the project. You can skip them by pressing the enter key.

After completing the process, the package.json file will have the following content:

```
{
  "name": "react-demo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

1.3. Installing Webpack

Webpack is a module bundler that allows us to bundle our project files into a single file.

```
npm install webpack webpack-cli --save-dev
```

The above command will add **webpack** and **webpack-cli** as dev dependencies to the project.

1.4. Installing React

Install **react** and **react-dom** as dependencies.

```
npm install react react-dom --save
```

1.5. Installing Babel

To make React work, we need to install **Babel** alongside it. We need Babel to transpile ES6 and JSX code into ES5.

Install **babel-core**, **babel-loader**, **babel-preset-env**, **babel-preset-react** as dev dependencies.

```
npm install @babel/core babel-loader @babel/preset-env @babel/preset-react --save-dev
```

- **babel-core**: Transforms ES6 code into ES5.
- **babel-loader**: Helps webpack to compile the code with preset settings.
- **babel-preset-env**: Preset that helps Babel to transform ES6, ES7, and ES8 code into ES5.
- **babel-preset-react**: Preset that helps to transform JSX code into JavaScript.

1.6. Creating the index file

index.js

Create an index.js file inside the src root folder. This file will be the entry point of our application. /src/index.js:

```
console.log("Đây là index.js");
```

index.html

Create an index.html file inside the src root folder. It should have the following content: /src/index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>React Demo</title>
</head>

<body>
    <div id="root">

    </div>
</body>

</html>
```

1.7. Entry and output files

Create a **webpack.config.js** file in the root directory of the project to define rules for our loaders.

Specify the entry point and output directory of our application inside the **webpack.config.js** file.

```
const path = require("path");
module.exports = {
  entry: "./src/index.js",
  output: {
    path: path.join(__dirname, "/dist"),
    filename: "index_bundle.js"
  }
}
```

In the above code snippet, Webpack will bundle all our JavaScript files into an index-bundle.js file inside the dist directory.

1.8. Webpack Loaders

Now add some loaders inside this file, which will be responsible for loading and bundling source files.

The content of the **webpack.config.js file** will be as follows:

```
const path = require("path");

module.exports = {
  entry: "./src/index.js",
  output: {
    path: path.join(__dirname, "/dist"),
    filename: "index_bundle.js"
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: "babel-loader"
        },
      },
      {
        test: /\.css$/,
        use: ["style-loader", "css-loader"]
      }
    ]
  }
};
```

Here, babel-loader is used to load our JSX/JavaScript files, css-loader is used to load and bundle all CSS files into one file, and style-loader will add all styles inside the document's head.

Before Webpack can use the css-loader and style-loader, we need to install them:

```
npm install css-loader style-loader --save-dev
```

1.9. .babelrc

Now, create a .babelrc file inside the root directory of the project with the following content:

```
{

  "presets": ["@babel/preset-env", "@babel/preset-react"]

}
```

This file will let Babel know which presets to use for transpiling the code. Here, we are using two presets:

- **env**: This preset is used to transpile ES6/ES7/ES8 code into ES5.
- **react**: This preset is used to transpile JSX code into ES5.

1.10. Compiling the files using Webpack

Add the following two lines to the scripts section in the **package.json** file:

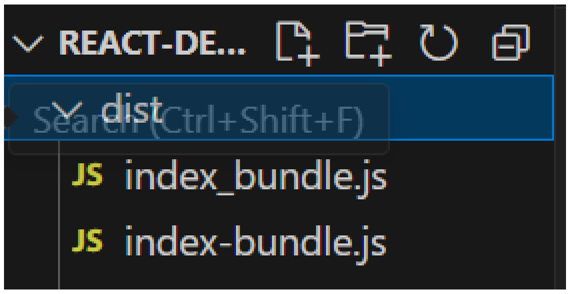
```
"start": "webpack --mode development --watch",

"build": "webpack --mode production"
```

Here, I have used watch, so whenever there are any changes in the source files, webpack will automatically compile all the source files.

Now, you can compile the project using the following command:

```
npm start
```

After running the above command, you will see an index_bundle.js file created in the /dist folder, which will contain the transpiled ES5 code from the index.js file.



```
index_bundle.js
index-bundle.js
```

1.11. App.js and App.css

Create an **App.js** file inside the components folder of src with the following content:

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

import '../styles/App.css';

class App extends Component {
    render() {
        return (
            <div>
                <h1>My React App!</h1>
            </div>
        );
    }
}

export default App;
```

Create an **App.css** file inside the styles folder of src with the following content:

```
h1 {
    color: blue;
    text-align: center;
    font-size: 40px;
}
```

This CSS file is used to ensure that the css-loader and style-loader work correctly.

Now, let's modify the index.js file with the following content:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./components/App.js";

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

1.12. Installing Html-webpack-plugin

Now, we also need to install html-webpack-plugin, which creates an HTML file, injects the script inside the HTML file, and writes this file to dist/index.html.

```
npm install html-webpack-plugin --save-dev
```

Now, we need to configure this plugin inside the webpack.config.js file:

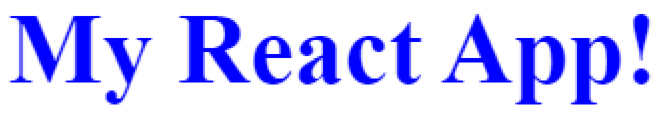
```
const path = require("path");
const HtmlWebpackPlugin = require("html-webpack-plugin");

module.exports = {
  entry: "./src/index.js",
  output: {
    path: path.join(__dirname, "/dist"),
    filename: "index-bundle.js"
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: ["babel-loader"]
      },
      {
        test: /\.css$/,
        use: ["style-loader", "css-loader"]
      }
    ]
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: "./src/index.html"
    })
  ]
};
```

Here, the value of template is the index.html file that we created earlier, and it uses this file as a template and creates a new file named index.html inside the /dist folder with the injected script.

The setup process is almost complete, all we have to do is compile the source files using webpack. You can run the project with the following command:

```
npm start
```

You will get the output inside the /dist folder of the project. Now, open index.html in a web browser, and you will see the content as 'My React App!'



My React App!

However, this method has a drawback that you have to manually refresh the webpage to see the changes you made. To have webpack watch our changes and refresh the webpage whenever any changes are made to our components, we can install webpack-dev-server.

1.13. Installing Webpack-dev-server

Install webpack-dev-server with the following command:

```
npm install webpack-dev-server --save-dev
```

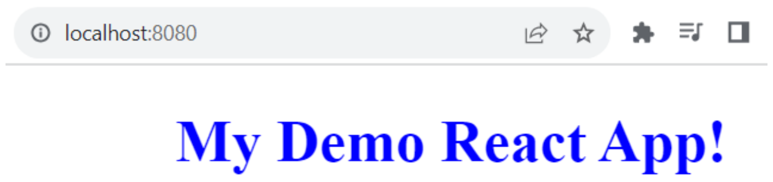
and then change the start in the scripts section of the package.json file:

```
"start": "webpack-dev-server --mode development --open --hot"
```

I have added --open and --hot to open and refresh the webpage whenever any changes are made to the components.

Now, open the terminal and run the following command:

```
npm start
```



My Demo React App!

2. Write JSX and ES6 code

2.1. Write JSX code:

- Create a simple React component called "HelloWorld" that displays the text "Hello, World!".
- Use JSX to create the component and display the content.

```
import React from 'react';
function HelloWorld() {
  return <div>Hello, World!</div>;
}
export default HelloWorld;
```

2.2. Write ES6 code:

- Create a simple ES6 class called "Person" with two properties: "name" and "age".
- Use ES6 syntax to initialize and access the properties of the class.

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  sayHello() {
    console.log(`Hello, my name is ${this.name} and I am ${this.age}
years old.`);
  }
}

const person = new Person("John", 25);
person.sayHello();
```