

NVM, React and Vite

Compiled by Tan Bee Hoon (contact: tbeehoon@gmail)

This readme shows:

1. Task1 - Explain React Core Features and its advantages.
2. Task2 - **Set Up a Basic React Environment**
3. Task3 - **Create and Render Functional Components**
4. Task4 - **Use JSX for Structuring Components**
5. How to set up the environment for NVM, React and Vite.

1. Task1 - Explain React Core Features and its advantages

1.1 React Core Features

The following is a brief explanation of React's core features: component-based architecture, Virtual DOM, and unidirectional data flow

1.1.1 Component-Based Architecture

- React applications are built using reusable, self-contained components.
- component encapsulates its own structure, logic, and style, making code modular and easier to maintain.
- Components can be nested, managed, and reused throughout the application.
- Components accept **props** (inputs) and manage internal **state**, then compose together like Lego bricks to form whole pages.
- This keeps concerns local and makes reuse easy.

1.1.2 Virtual DOM

- React uses a Virtual DOM, which is a lightweight copy of the actual DOM.
- When the state of an object changes, React updates the Virtual DOM first, then efficiently updates only the changed parts in the real DOM.
- When state changes, React computes the minimal set of real DOM updates via a diff ("reconciliation") and applies them efficiently, often batching multiple updates.
- This approach improves performance, especially in large and dynamic applications.

1.1.3 Unidirectional Data Flow

- Data in React flows in a single direction, from parent to child components via props.
- State changes trigger re-renders that propagate downward.
- This single direction makes data paths explicit and easier to trace.
- This makes the data flow predictable and easier to debug, as changes in the application state are

managed in a controlled way.

1.2 Advantages

These features collectively enable developers to build scalable, high-performance, and maintainable web applications.

- **Maintainability:** The component-based structure allows developers to break down complex UIs into smaller, manageable pieces, making code easier to read, test, and maintain.
 - **Performance:** The Virtual DOM minimizes direct manipulation of the real DOM, resulting in faster updates and a smoother user experience.
 - **Predictability:** Unidirectional data flow ensures that data changes are predictable and traceable, reducing bugs and making applications easier to debug.
 - **Reusability:** Components can be reused across different parts of an application or even in different projects, speeding up development and ensuring consistency.
-

2. Task 2 - Set Up a Basic React Environment

Required task details:

a) Create an HTML file and include CDN links for React and ReactDOM.

b) Add a

with an id of "root" in the HTML body.

c) Write a simple React component inside a `tag and use ReactDOM.render() to render it to the page.`

The codes for the above tasks:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>React CDN Example</title>
  <!-- React and ReactDOM CDN links -->
  <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js">
</script>
  <!-- Babel for JSX support in the browser, just for this example task-->
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <!-- Root div for React -->
  <div id="root"></div>
  <!-- React code -->
  <script type="text/babel">
```

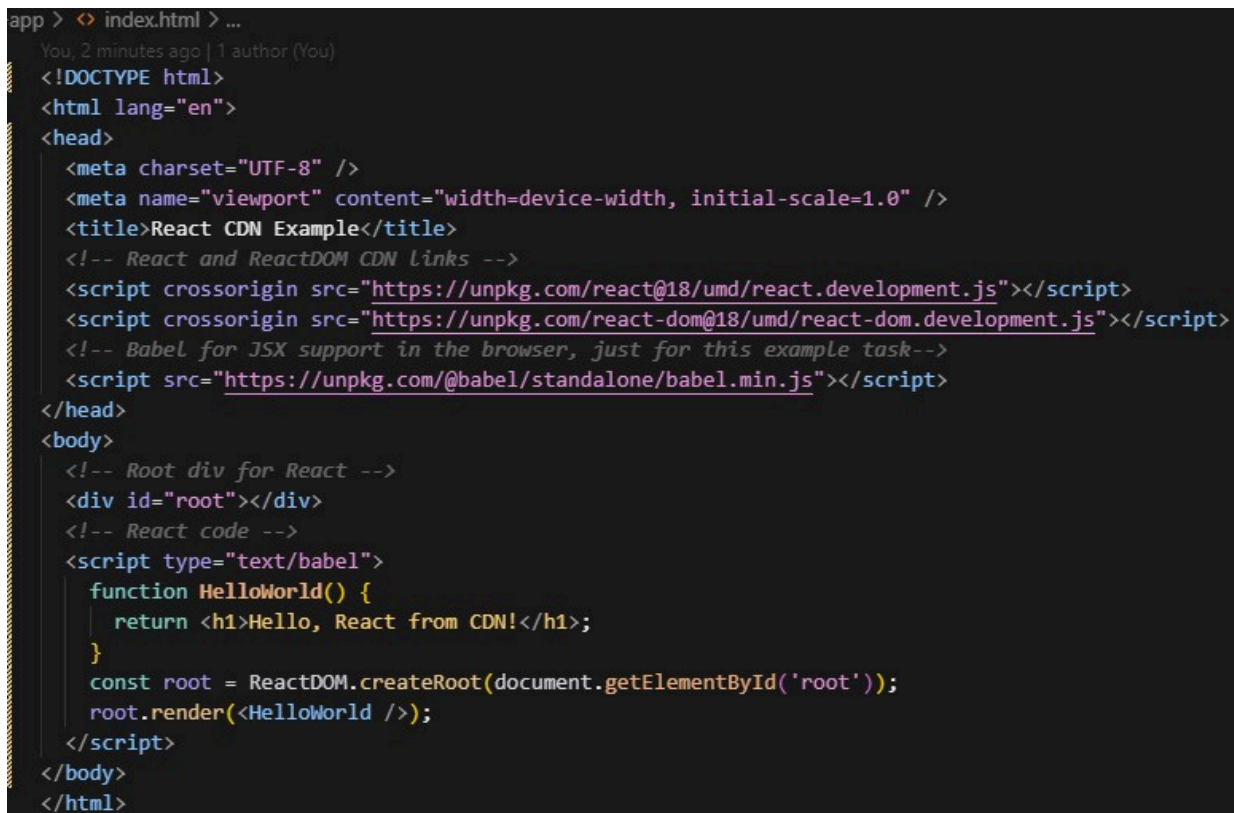
```

function HelloWorld() {
  return <h1>Hello, React from CDN!</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<HelloWorld />);
</script>
</body>
</html>

```

Screen captures for the above code:

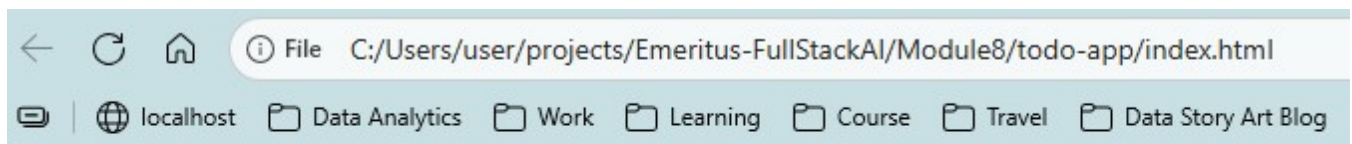


```

app > index.html > ...
You, 2 minutes ago | 1 author (You)
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>React CDN Example</title>
  <!-- React and ReactDOM CDN Links -->
  <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
  <!-- Babel for JSX support in the browser, just for this example task -->
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <!-- Root div for React -->
  <div id="root"></div>
  <!-- React code -->
  <script type="text/babel">
    function HelloWorld() {
      return <h1>Hello, React from CDN!</h1>;
    }
    const root = ReactDOM.createRoot(document.getElementById('root'));
    root.render(<HelloWorld />);
  </script>
</body>
</html>

```

Screen captures for the above in browser:



Hello, React from CDN!

3. Task 3 - Create and Render Functional Components

Required task details:

- Define a functional React component (e.g., GreetingComponent) that returns a simple JSX element.
- Render the component using ReactDOM.render() to display it on the page.

Important

Note: The local machine setup is using **React version 18**. Thus, not able to use **ReactDOM.render()**. Instead, **React version 18** and above with Vite, expect to use **createRoot**. Thus, part (b) of the task is modified slightly to use the supported **createRoot(...).render(...)**

The 3 files are modified to complete the task:

a. **GreetingComponent**: Created as a separate file (src/GreetingComponent.jsx) that returns a simple JSX element.

```
app > src > GreetingComponent.jsx > ...
You, 19 minutes ago | 1 author (You)
function GreetingComponent() {
  return <h2>Hello from GreetingComponent!</h2>;
}

export default GreetingComponent;
| Ctrl+L to chat, Ctrl+K to generate
```

b. **main.jsx**: Imports and renders GreetingComponent instead of App. Uses **createRoot(...).render(...)** instead of **ReactDOM.render()**.

```
app > src > main.jsx
import React from 'react'
import GreetingComponent from './GreetingComponent'

createRoot(document.getElementById('root')).render(
  <GreetingComponent />
)
```

c. **index.html**: Matches the Vite template (no Babel or CDN scripts, add a root div and a module script for main.jsx).

```
app > index.html > ...
You, 24 minutes ago | 1 author (You)
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Component-Based Architecture</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

d. run the command "**npm run dev**"

```
user@MSI MINGW64 ~/projects/Emeritus-FullStackAI/Module8/todo-app (main)
$ npm run dev

> todo-app@0.0.0 dev
> vite

VITE v7.1.4 ready in 355 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```

e. check the browser



4. Task 4 - Use JSX for Structuring Components

Required task details:

- a) Create a component that renders a list of items (using an unordered list) and a heading (e.g., "My To-Do List").
- b) The component should display at least three items in the list using JSX.

The 4 files are modified to complete the task:

- a. **GreetingComponent:** using the same file (src/GreetingComponent.jsx) that returns a simple JSX element with slight modification.
- b. **ToDoList:** create a new file (src/ToDoList.jsx) that contains the ToDoList component and exports it as default. Also, use React-Bootstrap components for layout.

```

app > src > TodoList.jsx > ...
function TodoList() {
  return (
    <div className="todo-list-left">
      <h3>My To-Do List</h3>
      <ul className="todo-list-ul-left">
        <li>Learn React basics</li>
        <li>Practice chinese painting</li>
        <li>Bake some bread</li>
      </ul>
    </div>
  );
}

export default TodoList;

```

c. **App.jsx**: Imports and renders both GreetingComponent and TodoList.

```

app > src > App.jsx > ...
You, 1 second ago | 1 author (You)
import GreetingComponent from './GreetingComponent'
import TodoList from './TodoList'
import './App.css'

function App() {
  return (
    <>
      <GreetingComponent />
      <TodoList />
    </>
  )
}

export default App

```

d. **main.jsx**: renders only the App component as the root of your application.

This makes App the main entry point for the UI, and all other components are organized and rendered through it.

Also the import of Bootstrap CSS is in src/main.jsx at the very top. This makes Bootstrap styles available globally in the app.

```

app > src > main.jsx
You, 2 minutes ago | 1 author (You)
import 'bootstrap/dist/css/bootstrap.min.css';
import React from 'react'
import { createRoot } from 'react-dom/client'
import App from './App'

createRoot(document.getElementById('root')).render(
  <App />
)

```

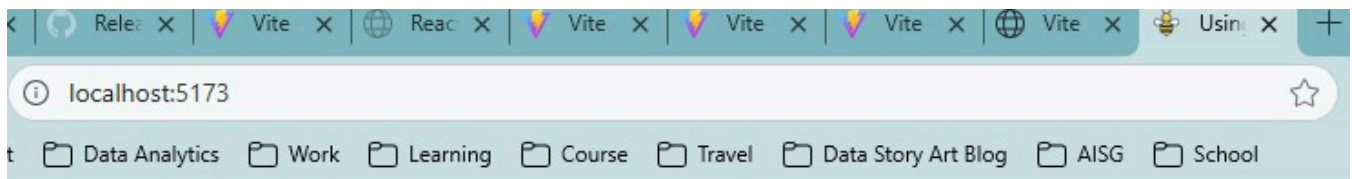
e. **index.html**: Not much change the in this file, only slight change to update the title and the favicon


```

app > <> index.html > ...
You, 5 minutes ago | 1 author (You)
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/png" href="/bee-icon.png" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Using JSX for Structuring Components</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>

```

f. run the command "**npm run dev**" and check output.



g. Final git push for todolist app is pushed to the following github repo:

<https://github.com/tbeehoon/todo-app/tree/main>

5. How to set up the environment

5.1 Install NVM (Node Version Manager)

Download the latest `nvm-setup.exe` from the releases page: <https://github.com/coreybutler/nvm-windows/releases>

Run the installer, then open a new PowerShell/Command Prompt.

Verify:

```
nvm -version
```

Tip

Avoid installing the “global” Node.js from nodejs.org if using NVM.

5.2 Install Node.js via NVM and set a default

Install the version required (LTS recommended), then make it the default so new terminals pick it automatically.

In PowerShell/Command Prompt, do the following installation.

```
# Install latest LTS
nvm install --lts

# OR install a specific version
nvm install 20

# Use it now
nvm use 20

# Make it the default for all new shells
nvm alias default 20
```

In bash, do verification.

```
# Verify
node -v
npm -v
```

Tip

Using bash instead because PowerShell/Command Prompt may not have the execution right for node.

5.3 Create a new React app with Vite

From any workspace folder in your terminal:

```
npm create vite@latest my-app -- --template react
# For TypeScript:
# npm create vite@latest my-app -- --template react-ts
```

Then install dependencies and run the dev server:

```
cd my-app
npm install
npm run dev
# vite typically starts at http://localhost:5173
```


💡 Tip

Ctrl-C to stop

5.4 Add Bootstrap to the React project

Install Bootstrap and its dependencies:

```
npm install bootstrap react-bootstrap
```

Import Bootstrap styles in `src/main.jsx` (or `src/main.tsx`` for TypeScript):

```
import 'bootstrap/dist/css/bootstrap.min.css'
```

Ready to use Bootstrap classes and React-Bootstrap components in app.

Example in `App.jsx`:

```
import Button from 'react-bootstrap/Button'

function App() {
  return (
    <div className="p-4">
      <h1>Hello, Bootstrap + React + Vite!</h1>
      <Button variant="primary">Click Me</Button>
    </div>
  )
}

export default App
```

5.5 Initialize Git

Version control the project using Git.

```
# Initialize a git repository
git init

# Add all project files
git add .

# Commit the files
git commit -m "Initial commit: setup React + Vite project"
```

To add to Github.

```
# Add remote
git remote add origin https://github.com/username/my-app.git

# Push changes
git branch -M main
git push -u origin main
```

Tip

In case identity need to be authenticated:

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

5.6 Setup .gitignore

Add a `.gitignore` file in the root of the project to exclude files and folders not required in version control. Some examples of items to include:

```
# dependencies
/node_modules

# production build
/dist

# logs
npm-debug.log*
*.log

# environment variables
.env
.env.local
.env.*.local

# IDE/editor folders
.vscode/
.DS_Store

# vite cache
.vite/
```

@Q.E.D.