Diploma Web Appication Development: Introduction

ICT50220 Diploma of Information Technology(Front-End Web Development)

Code	Title
ICTWEB517	Create web-based programs
ICTWEB546	Validate application design against specifications

North Metropolitan

ICT40120 Certificate IV in Programming

Web Application Development – Introduction to React

What is React?

- JavaScript library for building the user interface (UI)
- Involves the creation & reuse of components.
- created by Meta (for Facebook)
- originally single-page applications (SPA's) - NextJS is changing this
- native mobile apps

Use case: managing dynamic data.

Why use React?

- we describe what the UI should look like (declarative)
- not how to change the UI.
- we build encapsulated components (OOP)
- components manage their state (how they change)

Use case: admin dashboards

```
import React from 'react';


function Sidebar() {
  return (
    <div>
      <ul>
        <li>Dashboard</li>
        <li>Users</li>
        <li>Settings</li>
      </ul>
    </div>
  );
}
```

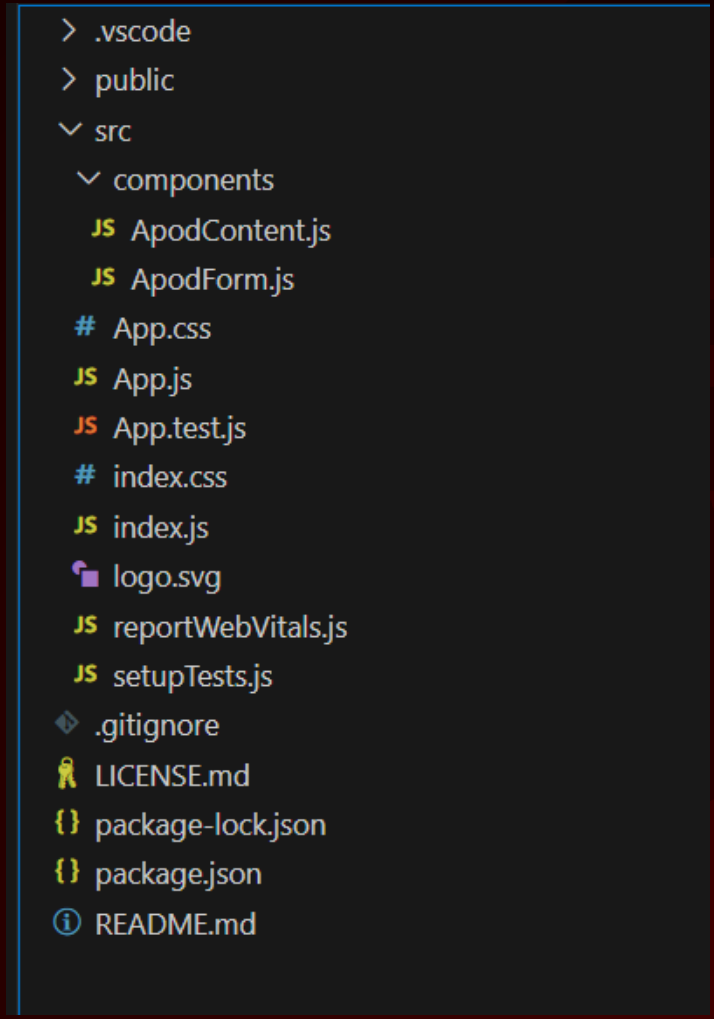
The React Learning Curve

- the React ecosystem including tools for state management, routing for navigation, UI components & more
- there is a massive community that contributes
- however, the use packages and plugins can be complex & hard to manage

```
// Core React
import React, { useState, useEffect } from 'react';

// Routing
import { BrowserRouter as Router, Route, Switch,
Link } from 'react-router-dom';

// State Management
import { Provider, useSelector, useDispatch } from
'react-redux';
import store from './store';
import { connect } from 'react-redux';

// UI Components (Material-UI as an example)
import Button from '@material-ui/core/Button';
import AppBar from '@material-ui/core/AppBar';
```

The React Development Environment

- use the CLI to start a new React application.
- this sets up your development environment
- lays out the initial structure & tooling for the project
- includes Webpack, Babel & ESLint.



Package Management – *package.json*

Source of truth for your project storing metadata for the project.

- manages the project's dependencies (production & development).
- script commands for repetitive tasks like building, testing & starting
- version control to manage conflicts as libraries/plugins change

```
{
  "name": "apod-app",
  "version": "1.0.0",
  "private": true,
  "dependencies": {
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-scripts": "4.0.3"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "my-custom-script": "run something clever"
  }
}
```

Package Managers – NPM & Yarn

- used to install, share & manage library dependencies
- manages React itself, React-DOM, and other libraries
- controls dependencies using the ***package-lock.json***

Hint: npx is included with npm – it allows you to run & test versions of packages without installing them on your global system

```
npm install <package-name>

npx install <package-name>

yarn install <package-name>
```

Documentation | Yarn
Documentation | NPM

Package Versions

- Minor & Patch versions are backward compatible
- Major versions may break previous code
- Caret(^): ^17.0.2 allows updates to any 17.x.x release, but not to 18.0.0 +
- “react-scripts”: “4.0.3” means the package version is set to this version only

```
"dependencies": {
      // Major.Minor.Patch
      "react": "^17.0.2",
      "react-dom": "^17.0.2",
      "react-scripts": "4.0.3"
},
```

Hint: read the release notes before installing major versions

React Tooling – CI/CD

Jest

- testing platform for functions, components, objects & can be automated in deployment

React Testing

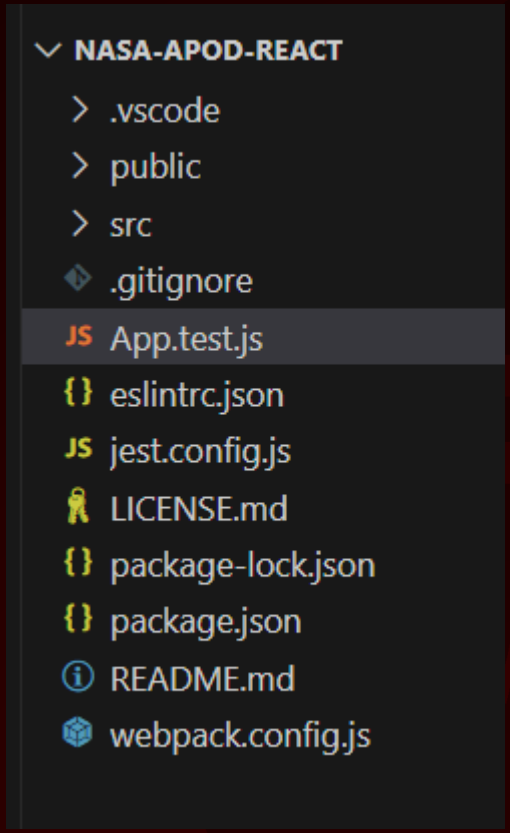
- use actual DOM nodes, simulating users interacting with your application.

ESLint

- enforces the style guide & can be automated in deployment

Webpack

- bundles your code & assets
- provides Hot Module Replacement



JSX

- JSX is an extension for JavaScript
- looks like HTML but is closer to JavaScript.
- we can include JavaScript expressions in JSX by wrapping JS in curly braces.
- JSX tags can be HTML tags **or** React components.

```
<div className="container">
      <h1 className="title">NASA Astronomy Picture of the Day</h1>
      <ApodForm fetchApodData={fetchApodData} />
      {error && <p className="has-text-danger">{error}</p>}
      <ApodContent apodData={apodData} />
</div>
```

JSX - Rules

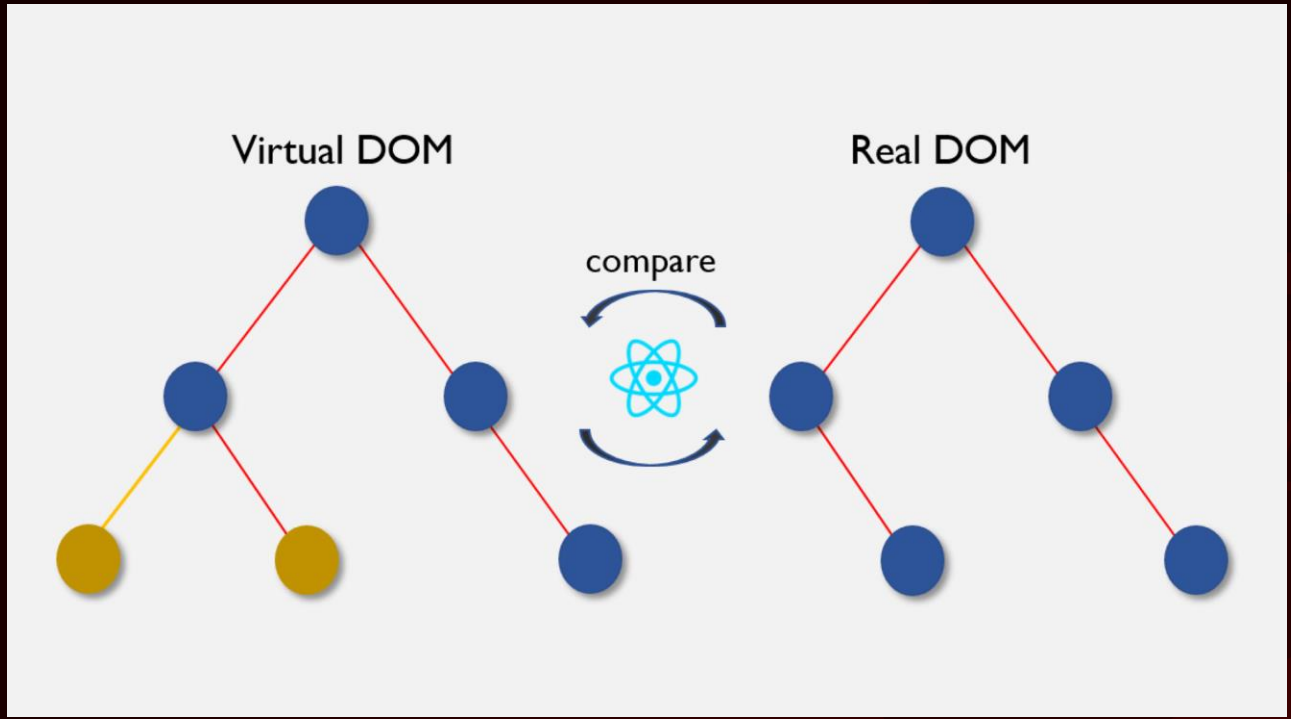
1. only return one element can be returned
2. we can however return a ‘fragment’ with multiple elements inside
3. we must use PascalCase for out components by convention
4. all elements/fragments must have a closing tags

```
// A fragment created by empty tags
<>
        <div className="container">
                <h1 className="title">NASA Astronomy Picture of the Day</h1>
                // PascalCase component name - ApodForm
                <ApodForm fetchApodData={fetchApodData} />
                {error && <p className="has-text-danger">{error}</p>}
                <ApodContent apodData={apodData} />
        </div>
// The closing tag for the fragment
</>
```

React & the DOM

React creates its own copy of the DOM (virtual)

- it compares the virtual & real DOM's
- it then re-renders only the changes (diffing) to the real DOM
- making it extremely fast



https://hackernoon.com/the-react-virtual-dom-explained

JSX uses React.createElement

- createElement converts elements written in JSX into vanilla JavaScript
- prepares the element representation without touching the actual DOM

```
// JSX
const element = <h1 className="greeting">Hello, world!</h1>;

// Translated to JS
const element = React.createElement(
  'h1',
  { className: 'greeting' },
  'Hello, world!'
);
```

ReactDOM

- manages the interaction between React components and the actual DOM.
- allows React to update only parts of the DOM that have changed (diffs)

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

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

Converting JSX with Babel

- Babel is a compiler that lets you use modern JavaScript (ES6+)
- Babel compiles JSX into **React.createElement()** code for our browser to use
- Specifically, it uses **@babel/preset-react**

```
{
    "presets": [
      "@babel/preset-env",
      "@babel/preset-react"
    ],
    "plugins": [
      // Example of a plugin
      "@babel/plugin-proposal-class-properties"
    ]
 }
```

Converting JSX with Babel

```
// JSX
const ApodDisplay = ({ title, imageUrl }) => (
  <div>
    <h1>{title}</h1>
    <img src={imageUrl} alt={title} />
  </div>
);
```

```
// Converts to the following in the DOM
"use strict";

const ApodDisplay = ({ title, imageUrl }) =>
React.createElement(
  "div",
  null,
  React.createElement(
    "h1",
    null,
    title
  ),
  React.createElement("img", { src: imageUrl, alt: title })
);
```

Challenge 1: Babel Setup

- initialise a new React project
- configure Babel
- create a simple React Greeting component
- ensure it compiles correctly using babel

Challenge 2

- create a JSX component that takes the array of items
- this is returned to the App component
- the component renders them in an ordered list.

Challenge 3

- update the JavaScript challenge 3 starter code to use React.createElement to render the DOM

Challenge 4

- update the React.createElement code you created in challenge 3 code to use JSX & Babel together.
- create a UserProfile component returns the user object detail to the App component to render the DOM.

Complete Session 07 - 01 - Tutorial - JSX