**NamasteDev**

Assignment 3 – Laying the Foundation

Q1: What is JSX?

The syntax of React.createElement(type, props, …children) is not developer friendly when it comes to nesting tags. As an alternative, Facebook developed JSX.

JSX or javascript XML is an extension of Javascript that lets you write HTML-like markup inside a Javascript file.

Ex: const element = <p>Learning React – Assignment 3</p>

The above statement is neither HTML nor a String. It might look like you are writing HTML inside JS, but that is not the case. This is an HTML-like (or XML-like in some cases) markup inside a JS file.

It is worth mentioning that JSX is not a requirement for using React. Each JSX element is the syntactic sugar for calling React.createElement() itself. So, anything that you do in JSX can be done in plain Javascript as well.

Reference: <https://legacy.reactjs.org/docs/react-without-jsx.html>

<https://legacy.reactjs.org/docs/introducing-jsx.html>

Q2: Superpowers of JSX

1. Putting Markup inside JS – For many years, markup content was kept in HTML, design in CSS and all the logic in JS files. All these were in separate files. But as web became more responsive and interactive, the logic started deciding the content markup. In other words, Javascript was in charge of HTML.
2. Specifying attributes in JSX – Since JSX is closer to JS, React DOM uses camelCase naming convention instead of HTML attribute names.

Ex: className instead of class(this is a keyword too)

tabIndex instead of tabindex

1. Embedding JS expressions in JSX – Any valid javascript can be injected into JSX using {}

const time = “morning”;

const message = <h2>Good {time}, How are you?</h2>

After compilation, JSX expressions are transpiled (by Babel) into React.createElement() which is nothing but a react element which in turn are our JS objects. Therefore, we can use JSX inside if, inside for loops, assign it to variables, accept it as arguments, return it from functions.

1. JSX prevents injection attacks -

Cross-Site Scripting(XSS) is a security vulnerability that allows an attacker to inject some malicious code into the client side webpage. This code is then unknowingly executed by the victim and lets the attacker rob sensitive user information.

The browsers cannot differentiate between the correct code and malicious code because the malicious code often includes Javascript along with some HTML, Flash that the browsers can execute.

3 types – Stored XSS attack(Persistent)

Reflected(non-persistent)

DOM-based XSS

<https://legacy.reactjs.org/docs/introducing-jsx.html>

<https://react.dev/learn/writing-markup-with-jsx>

<https://www.stackhawk.com/blog/react-xss-guide-examples-and-prevention/>

Q3: Role of type attribute in script tag? What options can I use there?

The “type” attribute in <script> tag specifies the content-type(or media-type or MIME-type) of the referenced script. This attribute is optional with a default value of “text/javascript”.

Various options:

1. type=”module” – This value causes the file to be treated as a module. The processing of the file content is deferred.

Use case – when trying to load a react file, error is displayed “Browser scripts cannot have imports/exports” . This is because browser expects normal js files for execution. It does not understand modules by default. To resolve this, we have to specify that the file is a module, now you can have imports/exports and load react files.

1. type=”importmap” – this value indicates that the body of the element contains an import map.
2. type=”specificationrules” – This value indicates that the body of the element contains specification rules.
3. type=”text/typescript” – script is written in typescript

Q4: {TitleComponent} vs {<TitleComponent/>} vs {< TitleComponent >< /TitleComponent >} in JSX

These are all different ways of rendering a component.

• {TitleComponent}: This value describes the TitleComponent as a javascript expression or a variable. The {} can embed a javascript expression or a variable inside it.

• <TitleComponent/> : This value represents a Component that is basically returning Some JSX value. In simple terms TitleComponent is a function that is returning a JSX value. A component is written inside the {< />} expression.

• <TitleComponent></TitleComponent> : <TitleComponent /> and <TitleComponent></TitleComponent> are equivalent only when < TitleComponent /> has no child components. The opening and closing tags are created to include the child components.

Example:

<TitleComponent>

<FirstChildComponent />

<SecondChildComponent />

<ThirdChildComponent />

</TitleComponent>

Notes:

Babel transpiles jsx to react code that is understood by the js engine/browser