Module 18 – React-JS Full Stack Assignment (Theory)

1. Introduction to React.js

Q1: What is React.js? How is it different from other JavaScript frameworks and libraries?
Answer:
React.js is a JavaScript **library** used for building user interfaces, mainly for single-page applications. It focuses only on the **view layer** of the app.

Key Differences:

1. **Library, not a framework** – React handles only the UI; frameworks like Angular handle more (routing, services).
2. **More flexible** – You can choose your own tools and libraries.
3. **Easier to start** – Especially if you know JavaScript, but learning the full ecosystem takes time.
4. **Faster updates** – Uses Virtual DOM to make updates efficient.
5. **Big community** – Lots of support, tutorials, and tools available.

Q2: What are the core principles of React, like the Virtual DOM and component-based architecture?
Answer:
React is based on the following main ideas:

1. **Component-Based** – The UI is made up of small, reusable pieces called components.
2. **Virtual DOM** – React updates a lightweight copy of the DOM first, then updates the real DOM more efficiently.
3. **One-way Data Flow** – Data flows from parent to child only, which makes things predictable.

4. **JSX Syntax** – Allows writing HTML-like code inside JavaScript, which makes UI code easier to read and write.

Q3: What are the benefits of using React.js in web development?
Answer:

1. **Reusable Components** – Avoids repeating code.
2. **High Performance** – Thanks to Virtual DOM.
3. **Easy Data Handling** – One-way data flow simplifies debugging.
4. **Big Ecosystem** – Tools like Redux and React Router are easy to use.
5. **Strong Community** – Many online resources are available.
6. **SEO-Friendly** – Can render content on the server.
7. **Cross-Platform Support** – Use React Native to build mobile apps too.

---

2. JSX (JavaScript XML)

Q1: What is JSX in React.js? Why is it used?
Answer:
JSX is a syntax that lets you write HTML-like code inside JavaScript.

Why it's used:

1. **Makes code clearer** – You can see the UI layout inside JS.
2. **Declarative** – You write what the UI should look like.
3. **Mix JS with HTML** – You can add dynamic values using JS expressions.
4. **Gets compiled** – JSX is converted into regular JS code that React understands.

Q2: How is JSX different from regular JavaScript? Can we use JS inside JSX?
Answer:
Differences:

1. JSX looks like HTML but is not valid JavaScript.
2. It needs to be converted into JS using tools like Babel.

Yes, you can use JavaScript inside JSX by wrapping expressions in curly braces `{}`.

Example:

```
const name = "King";
return <h1>Hello, {name}</h1>;
```

Q3: Why do we use curly braces `{}` in JSX?
Answer:
Curly braces are used to include **JavaScript expressions** inside JSX.

Importance:

1. **Dynamic content** – Like variables and function results.
2. **Evaluates expressions** – e.g., math, string operations.
3. **Keeps logic and markup clean** – Helps manage code better.

---

3. Components (Functional & Class)

Q1: What are components in React? What's the difference between functional and class components?
Answer:
Components are the building blocks of a React app's UI.

Functional Components:

- Written as functions.
- Originally stateless, but can now use **Hooks** (like useState).

- Shorter and easier to read.

Class Components:

- Written using ES6 classes.
- Can manage their own **state** and use **lifecycle methods**.

Example of Functional Component:

```
const Welcome = () => <h1>Hello</h1>;
```

Example of Class Component:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello</h1>;
  }
}
```

Q2: How do you pass data to a component using props?
Answer:

Props (short for "properties") are used to pass data from a parent to a child component.

Example:

```
function Parent() {
  return <Child name="Alice" />;
}

function Child(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

You can also use **destructuring**:

```
function Child({ name }) {
  return <h1>Hello, {name}</h1>;
}
```

Q3: What is the role of the `render()` method in class components?
Answer:

- `render()` returns JSX that describes what the UI should show.
- React calls `render()` when state or props change.
- Must return **only one** parent element.

Example:

```
class Hello extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

---

4. Props and State

Q1: What are props in React.js? How are they different from state?
Answer:

Props	State
Passed from parent	Local to the component
Read-only	Can be changed (mutable)
Cannot be modified by the component	Modified using `useState` or `this.setState()`

Props are for **external data**, while state is for **internal data**.

Q2: What is state in React, and how is it used?
Answer:
State is data managed **inside** a component that can change over time.

In Functional Components (with useState):

```
import { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
  return <button onClick={() => setCount(count + 1)}>Count: {count}</button>;
}
```

In Class Components:

```
class Counter extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  }

  render() {
    return <button onClick={this.increment}>Count: {this.state.count}</button>;
  }
}
```

Q3: Why is `this.setState()` used in class components? How does it work?
Answer:

- It updates the component's **state**.
- Triggers a **re-render** to show new data.
- Does a **shallow merge** with the old state.
- Updates can be **asynchronous**, so it's better to use a function when relying on old state.

Example:

```
this.setState(prevState => ({
  count: prevState.count + 1
```

```
}));
```
