# Full stack Development Paper-I

# Unit – 1

## Getting Started with React.js

## Contents



## 1.1 Introduction

### Introduction to React and Its Benefits

**React** is a JavaScript library developed by Facebook for building user interfaces, particularly for single-page applications. It allows developers to create large web applications that can

update and render efficiently in response to data changes. React is known for its simplicity, scalability, and speed.

## Benefits of React:

1. **Component-Based Architecture**: React applications are built using components, which are independent, reusable pieces of code. This makes development more organized and manageable.

2. **Virtual DOM**: React uses a virtual DOM to efficiently update the actual DOM. When the state of a component changes, React updates the virtual DOM first, compares it with the previous version, and then updates only the changed parts of the real DOM.

3. **Declarative UI**: React allows developers to describe what the UI should look like for a given state, and it takes care of rendering the UI in the most efficient way.

4. **Unidirectional Data Flow**: React's data flow is one-way, making it easier to understand and debug. Data flows from parent to child components, ensuring that the application state is predictable.

5. **Rich Ecosystem**: React has a rich ecosystem with numerous libraries and tools that complement it, such as Redux for state management, React Router for navigation, and more.

## Environment Setup for React Development

To set up a development environment for React, follow these steps:

1. **Install Node.js and npm**:

   - Node.js: A JavaScript runtime built on Chrome's V8 JavaScript engine.

   - npm: Node Package Manager, used to install packages.

   - Download and install Node.js from [nodejs.org](https://nodejs.org/).

2. **Install a Code Editor**:

   - Visual Studio Code (VS Code) is highly recommended for its robust features and extensions tailored for JavaScript and React development.

   - Download and install VS Code from [code.visualstudio.com](https://code.visualstudio.com/).

3. **Install Create React App**:

   - Create React App is a tool to set up a modern React application without configuration.

   - Open your terminal and run:

```bash
npx create-react-app my-app
cd my-app
npm start
```

- This will create a new React project and start the development server.

## Refresher on ES6 Concepts

React uses modern JavaScript features introduced in ES6 (ECMAScript 2015) and later. Key ES6 concepts include:

1. **Arrow Functions**:

```javascript
const add = (a, b) => a + b;
```

2. **Template Literals**:

```javascript
const name = "World";
console.log(`Hello, ${name}!`);
```

3. **Destructuring**:

```javascript
const person = { name: "John", age: 30 };
const { name, age } = person;
```

4. **Modules**:

```javascript
// Exporting
export const greet = () => console.log("Hello!");

// Importing
import { greet } from './greet';
```

5. **Classes**:

```javascript
class Person {
  constructor(name) {
    this.name = name;
  }
  greet() {
    console.log(`Hello, ${this.name}!`);
  }
}
```

6. **Promises**:

```javascript
const fetchData = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => resolve("Data"), 1000);
  });
};
fetchData().then(data => console.log(data));
```

## Create React App

**Create React App** is an officially supported way to create single-page React applications. It offers a modern build setup with no configuration.

1. **Installation**:

```bash
npx create-react-app my-app
cd my-app
npm start
```

2. **Running the App**:

- Navigate to the project directory.

- Run `npm start` to start the development server.

- Open [http://localhost:3000](http://localhost:3000) to view it in the browser.

## Folder Structure

When you create a new React app using Create React App, it generates a folder structure like this:

```
my-app/
├── node_modules/
├── public/
│   ├── index.html
│   └── ...
├── src/
│   ├── App.css
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── logo.svg
│   └── ...
├── .gitignore
├── package.json
├── README.md
└── yarn.lock
```

### Key Folders and Files:

- `node_modules/`: Contains all the npm packages installed for the project.
- `public/`: Contains the public assets, including the HTML file where the React app mounts.
- `src/`: Contains the source code of the React application.
  - `App.js`: The main component.
  - `index.js`: The entry point of the React application.
- `.gitignore`: Specifies which files and directories to ignore in version control.
- `package.json`: Lists the project dependencies and scripts.
- `README.md`: Provides information about the project.

This setup provides a solid foundation to start building your React application efficiently.

## 1.2 Templating using JSX

JSX (JavaScript XML) is a syntax extension for JavaScript used with React to describe what the UI should look like. JSX may look like HTML, but it's transformed into JavaScript objects before being rendered in the browser.

### Understanding Component Architecture and Its Significance

In React, the user interface is built using components, which are the building blocks of a React application. Components make it easier to break down complex UIs into smaller, manageable pieces that can be reused and independently developed.

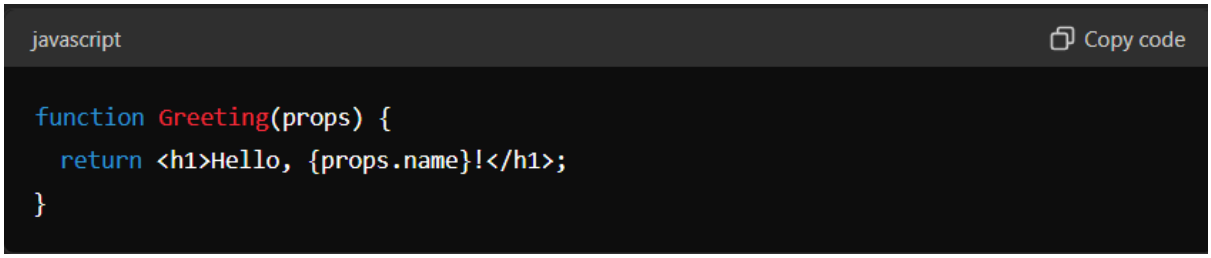
#### Significance of Component Architecture:

1. **Reusability**: Components can be reused across different parts of the application, reducing redundancy.
2. **Maintainability:** Breaking down the UI into components makes the code easier to read, maintain, and debug.
3. **Separation of Concerns:** Each component handles its own logic and rendering, promoting a clear separation of concerns.
4. **Testability:** Components can be tested individually, making the testing process more straightforward.

### Introduction to Components and Their Types

There are two main types of components in React:

1. **Functional Components:** These are simple JavaScript functions that accept props as an argument and return JSX.



2. **Class Components:** These are ES6 classes that extend React.Component and have a render method that returns JSX.

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

## Working with React.createElement to Create Elements

React.createElement is a method used to create React elements. JSX is syntactic sugar for React.createElement, which means the following JSX:

```jsx
const name = 'John';
const element = <h1>Hello, {name}!</h1>;
```

## Logical Operators in JSX

JSX supports logical operators like && for conditional rendering. If the condition is true, the element after && is rendered.

```jsx
const isLoggedIn = true;
const element = (
  <div>
    {isLoggedIn && <h1>Welcome back!</h1>}
  </div>
);
```

## Attributes in JSX

Attributes in JSX are written in camelCase, such as className instead of class, onClick instead of onclick, etc.

```jsx
const element = <button className="btn" onClick={handleClick}>Click Me</button>;
```

## Children in JSX

JSX can include children elements nested inside other elements.

```jsx
const element = (
  <div>
    <h1>Hello, world!</h1>
    <p>This is a paragraph.</p>
  </div>
);
```

**Example**

Here's an example that puts all these concepts together:

jsx Copy code

```jsx
import React from 'react';

// Functional Component
function UserGreeting(props) {
  return <h1>Welcome back, {props.name}!</h1>;
}

// Class Component
class GuestGreeting extends React.Component {
  render() {
    return <h1>Please sign up.</h1>;
  }
}

// Main App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = { isLoggedIn: false };
  }

  handleLogin = () => {
    this.setState({ isLoggedIn: true });
  };

  render() {
    const { isLoggedIn } = this.state;
    return (
      <div>
        {isLoggedIn ? <UserGreeting name="John" /> : <GuestGreeting />}
        <button onClick={this.handleLogin}>
          {isLoggedIn ? 'Logout' : 'Login'}
        </button>
      </div>
    );
  }
}

export default App;
```

In this example:

- We define a functional component UserGreeting that uses props.
- We define a class component GuestGreeting.
- The App component maintains the state and conditionally renders either UserGreeting or GuestGreeting based on the isLoggedIn state.
- JSX expressions, attributes, and logical operators are used to manage the rendering logic.

## 1.3 Working with Props and State :

### Understanding the Concept of State and Its Significance in React

State is a built-in React object used to hold data or information about the component. A component's state can change over time, usually as a result of user actions, and when it changes, the component re-renders to reflect the new state.

### Significance of State:

1. **Dynamic UIs:** State allows components to render dynamic content based on user interaction or other events.
2. **Component Lifecycles:** State is managed within the component, which makes it easier to handle complex component lifecycles.
3. **Encapsulation:** State is local to the component where it's defined, which encapsulates the component's behavior and data.

### Setting and Reading Component States

State is managed within class components using this.state and this.setState or with hooks like useState in functional components.

**Class Components:**

javascript Copy code

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

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

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

**Functional Components with Hooks:**

```javascript
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

## Working with Props to Pass Data Between Components

Props (short for properties) are read-only attributes used to pass data from parent components to child components. They help create dynamic and reusable components.

**Example:**

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

function App() {
  return <Greeting name="John" />;
}
```

In this example, the App component passes the name prop to the Greeting component.

## Validating Props Using PropTypes

PropTypes are used to enforce the type and presence of props a component receives, which helps catch bugs and improve code readability.

**Example:**

```javascript
import PropTypes from 'prop-types';

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

Greeting.propTypes = {
  name: PropTypes.string.isRequired,
};

function App() {
  return <Greeting name="John" />;
}
```

In this example, name is validated to be a string and is required for the Greeting component.

### Using Default Props to Supply Default Values

Default props are used to define default values for props when they are not provided by the parent component.

**Example:**

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

Greeting.defaultProps = {
  name: 'Guest',
};

function App() {
  return <Greeting />;
}
```

In this example, if name is not provided, it defaults to 'Guest'.

### Comprehensive Example

Here's a more comprehensive example that demonstrates state, props, PropTypes, and default props:

javascript Copy code

```javascript
import React, { useState } from 'react';
import PropTypes from 'prop-types';

function Counter({ initialCount }) {
  const [count, setCount] = useState(initialCount);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  );
}

Counter.propTypes = {
  initialCount: PropTypes.number,
};

Counter.defaultProps = {
  initialCount: 0,
};

function App() {
  return (
    <div>
      <h1>Counter App</h1>
      <Counter initialCount={5} />
    </div>
  );
}

export default App;
```

In this example:

- The Counter component uses the useState hook to manage the count state.
- The initialCount prop is passed from the App component and is used to initialize the count state.
- PropTypes ensure that initialCount is a number.
- Default props provide a default value for initialCount if it is not supplied by the parent component.

This setup allows for a robust and flexible React component architecture, enabling effective state management and clear data flow between components.