1. Installing Node.js and npm

- **Node.js**: A JavaScript runtime that allows you to run JavaScript outside the browser (server-side).
- **npm (Node Package Manager)**: A tool for installing and managing JavaScript libraries.

Steps to Install:

1. **Download Node.js** from https://nodejs.org (LTS version recommended).
2. **Verify Installation**:

```
node -v  # Check Node.js version
npm -v   # Check npm version
```

3. **Update npm** (optional):

```
npm install -g npm@latest
```

2. Introduction to Frameworks and Libraries

- **Framework** (e.g., Angular): Provides a full structure with strict rules.
- **Library** (e.g., React): Provides reusable functions/components but lets you decide architecture.

React as a Library:

- Focuses on the **view layer** (UI).
- Uses a **component-based** approach.

3. Introduction to React

- Developed by **Facebook**.
- Uses a **Virtual DOM** for efficient updates.
- Follows **unidirectional data flow** (parent → child).

Key Features:

1. **Component-Based Architecture**
   - Break UI into reusable components.
   - Example: `Button`, `Navbar`, `Card`.
2. **Virtual DOM**
   - A lightweight copy of the real DOM for performance optimization.
3. **Unidirectional Data Flow**
   - Data flows from parent to child via **props**.
   - State changes trigger re-renders.

4. Setting Up the Development Environment

Using `create-react-app` (CRA):

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

- Runs a dev server at `http://localhost:3000`.

Project Structure:

```
my-app/
├── node_modules/  # Dependencies
├── public/       # Static files (index.html)
├── src/          # React code
│   ├── App.js     # Main component
│   ├── index.js   # Entry point
│   └── ...
├── package.json   # Project config
└── ...
```

5. Creating a Basic React Project Structure

Example: `App.js`

```
import React from 'react';

function App() {
  return (
  <div>
    <h1>Hello React!</h1>
  </div>
 );
}
export default App;
```

6. Component-Based Architecture

- **Functional Components** (preferred with Hooks):

```
function Greeting({ name }) {
 return <h2>Hello, {name}!</h2>;
}
```

- **Class Components** (legacy):

```
class Greeting extends React.Component {
 render() {
  return <h2>Hello, {this.props.name}!</h2>;
 }
}
```

7. JSX Syntax

- JavaScript XML (JSX) allows HTML-like syntax in JavaScript.
- **Example**:

```
const element = <h1>Hello, JSX!</h1>;
```

- **Embedding Expressions**:

```
const name = "Alice";
const greeting = <p>Hello, {name}!</p>;
```

8. Component Lifecycle Methods (Class Components)

Method	Purpose
`componentDidMount()`	Runs after component renders (API calls).
`componentDidUpdate()`	Runs after component updates.

Method	Purpose
`componentWillUnmount()`	Runs before component removal (cleanup).

Example:

```
class MyComponent extends React.Component {
  componentDidMount() {
    console.log("Component mounted!");
  }
  render() {
    return <div>Lifecycle Example</div>;
  }
}
```

9. Handling Events in React

- Similar to DOM events but camelCased (`onClick` instead of `onclick`).
- **Example**:

```
function Button() {
  const handleClick = () => {
    alert("Button clicked!");
  };
  return <button onClick={handleClick}>Click Me</button>;
}
```

10. Managing Component State

Using `useState` (Functional Components)

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

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

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

Using `this.state` (Class Components)

```
class Counter extends React.Component {
  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>
    );
  }
}
```

Summary

Topic	Key Points
Node.js & npm	Runtime & package manager for React.
React Basics	Components, Virtual DOM, JSX.
State & Props	`useState`, `this.state`, passing data.
Events	`onClick`, `onChange`, etc.
Lifecycle	`componentDidMount`, `useEffect`.

Using Vite

Vite is a modern, fast build tool that provides a better development experience compared to traditional tools like `create-react-app`. It offers **instant server start**, **hot module replacement (HMR)**, and **optimized builds**.

Step 1: Install Node.js and npm

Before using Vite, ensure you have **Node.js (v14.18+ or v16+)** installed:

bash

```bash
node -v  # Check Node.js version
npm -v   # Check npm version
```

If not installed, download from https://nodejs.org.

Step 2: Create a React Project with Vite

Run the following command to scaffold a new React project:

```
npm create vite@latest my-react-app --template react
```

- `my-react-app` → Your project name.
- `--template react` → Specifies React as the framework.

Alternative (Using Yarn)

```
yarn create vite my-react-app --template react
```

Step 3: Navigate to the Project Directory

```
cd my-react-app
```

Step 4: Install Dependencies

```
npm install
```

or (if using Yarn):

Step 5: Start the Development Server

```
npm run dev
```

or (if using Yarn):

```
yarn dev
```

- This starts a dev server at `http://localhost:5173` (default port).

Step 6: Explore the Project Structure

```
my-react-app/
├── node_modules/   # Dependencies
├── public/         # Static assets (favicon, etc.)
├── src/            # React source code
│   ├── App.jsx     # Main React component
│   ├── main.jsx    # Entry point (renders App)
│   ├── index.css   # Global styles
│   └── ...
├── .gitignore      # Files to ignore in Git
├── index.html      # Root HTML file
├── package.json    # Project config & scripts
└── vite.config.js  # Vite configuration
```

Step 7: Modify `App.jsx` (Example)

Replace the default content in `src/App.jsx`:

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

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

  return (
    <div>
      <h1>Hello Vite + React!</h1>
      <button onClick={() => setCount(count + 1)}>
        Count: {count}
      </button>
    </div>
  );
}

export default App;
```

- The browser will **auto-reload** due to **HMR (Hot Module Replacement)**.

Step 8: Build for Production

```
npm run build
```

or (if using Yarn):

```
yarn build
```

- Generates optimized files in the `dist/` folder.

Step 9: Preview the Production Build

```
npm run preview
```

or (if using Yarn):

```
yarn preview
```

- Runs a local server to test the optimized build.

Why Use Vite Over `create-react-app` (CRA)?

Feature	Vite	CRA
Speed	⚡ Ultra-fast (ESM-based)	🐢 Slower (Webpack-based)
HMR (Hot Reload)	✅ Instant updates	⏳ Slower updates
Build Time	🚀 Optimized	🐌 Slower
Configuration	🔧 Flexible (easy to customize)	🔒 Limited (eject needed)