Prerequisites section

* Full Stack Overview
* Frontend Deep Dive (HTML, CSS, JS)
* Backend & Databases
* Key Tech Differences (Library vs Framework vs API)

**🌐 Prerequisites – What You Should Know Before React**

Before diving into React, it's good to have a basic understanding of **Full Stack Web Development**.

**🔁 Full Stack Development Overview**

**Full Stack** means working with both:

* **Frontend** (what users see)
* **Backend** (what powers the logic, and server)
* **Database** (what manipulate the data)

**🖼 Frontend Technologies**

|  |  |
| --- | --- |
| Tech | Description |
| HTML | Content and Structure of the webpage. Think of it like the **skeleton**. |
| CSS | Styling and layout. Adds **colors, spacing, and responsiveness**. |
| JavaScript | Makes the page **interactive** — for animations, validations, UI logic, etc. |

**💡 Example:**

<button onclick="alert('Hello!')">Click Me</button>

**🔧 Backend Technologies**

Backend runs **on the server**, handles **logic, processing, APIs**, and connects to the database.

Common backend languages:

* **Java** (Spring Boot)
* **Python** (Django, Flask)
* **Node.js, Express.js** (JavaScript backend)
* **PHP**, **.NET (C#)**, etc.

**🗃️ Databases**

Where we store data permanently.

| **Type** | **Examples** |
| --- | --- |
| **Relational (SQL)** | MySQL, PostgreSQL, Oracle |
| **Non-relational (NoSQL)** | MongoDB, Firebase |

**🎨 Deep Dive into Frontend**

**HTML – Structure**

* Defines headings, paragraphs, images, buttons, forms, etc.
* Tags like <h1>, <div>, <p>, <img>, <a>, <form>

**CSS – Styling & Layout**

* Adds design: colors, fonts, spacing
* Handles responsiveness using **media queries** and **flex/grid**

@media (max-width: 600px) {

.box {

flex-direction: column;

}

}

**JavaScript – Interaction**

* Handles logic and events (onClick, onSubmit)
* Can validate forms, change elements, interact with APIs

## 📘 JavaScript ES6+ Features You Should Know for React

### 1. ****Arrow Functions (****=>****)****

Arrow functions are a shorter syntax for writing functions, and they do **not bind their own this**, which is helpful in React components.

#### Example:

// Traditional function

function greet(name) {

return `Hello, ${name}`;

}

// Arrow function

const greet = (name) => `Hello, ${name}`;

#### In React:

const Button = ({ onClick }) => <button onClick={onClick}>Click me</button>;

### 2. let ****and**** const

* let is used for variables that can change.
* const is for constants (values that don’t change).

#### Example:

let count = 1;

count = 2;

const name = 'React';

// name = 'Vue'; // ❌ This will throw an error

In React, you'll often use const for defining components and let for variables that might change.

### 3. ****Template Literals (**** ****)****

Allows embedding variables inside strings using ${}.

#### Example:

const name = 'React';

console.log(`Welcome to ${name}!`);

In React:

const user = 'Alice';

return <p>Hello, {`${user}`}</p>;

### 4. ****Destructuring****

#### Arrays:

const nums = [1, 2];

const [a, b] = nums; // a = 1, b = 2

#### Objects:

const user = { name: 'John', age: 30 };

const { name, age } = user;

In React (very common for props):

const UserCard = ({ name, age }) => (

<div>{name} - {age}</div>

);

### 5. ****Spread (****...****) and Rest Operators****

#### Spread - to copy or merge arrays/objects:

const arr = [1, 2];

const newArr = [...arr, 3]; // [1, 2, 3]

const obj = { name: 'Alice' };

const newObj = { ...obj, age: 25 };

#### Rest - to gather remaining values:

function sum(...nums) {

return nums.reduce((a, b) => a + b);

}

In React (spread props):

const props = { name: 'Bob', age: 25 };

<UserCard {...props} />;

### 6. ****Default Parameters****

Provide default values to function parameters.

#### Example:

function greet(name = 'Guest') {

return `Hello, ${name}`;

}

In React:

const Welcome = ({ name = 'Guest' }) => <h1>Hi, {name}</h1>;

### 7. ****Classes and**** this

React supports both class and function components. Class components manage state and lifecycle methods.

#### Class Syntax:

class Person {

constructor(name) {

this.name = name;

}

greet() {

console.log(`Hi, I’m ${this.name}`);

}

}

In React (class component):

class MyComponent extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

Note: Function components with **hooks** are now preferred.

### 8. ****Modules (****import/export****)****

JavaScript modules allow you to split code into separate files and reuse them.

#### Exporting:

// utils.js

export const add = (a, b) => a + b;

#### Importing:

// app.js

import { add } from './utils';

React heavily relies on modules to organize components.

### 9. ****Promises & Async/Await****

Used to handle asynchronous operations like fetching data.

#### Promises:

fetch('https://api.example.com/data')

.then(res => res.json())

.then(data => console.log(data));

#### Async/Await (preferred syntax):

const fetchData = async () => {

const res = await fetch('https://api.example.com/data');

const data = await res.json();

console.log(data);

};

### 10. ****Array Methods (****map****,**** filter****,**** reduce****, etc.)****

#### .map() – transform arrays (used for rendering lists):

const names = ['Alice', 'Bob'];

const listItems = names.map(name => <li key={name}>{name}</li>);

#### .filter() – remove items based on condition:

const filtered = [1, 2, 3].filter(num => num > 1); // [2, 3]

#### .reduce() – accumulate values:

const sum = [1, 2, 3].reduce((acc, num) => acc + num, 0); // 6

## ✅ Summary Table

|  |  |  |
| --- | --- | --- |
| Feature | Used For | Example in React |
| Arrow Functions | Short syntax, auto-bind this | onClick={() => doSomething()} |
| let & const | Variable declaration | const MyComponent = () => {} |
| Template Literals | Interpolated strings | Hello, ${user} |
| Destructuring | Extract values from arrays/objects | const { name } = props; |
| Spread/Rest Operators | Copy/merge or gather args | <Component {...props} /> |
| Default Parameters | Provide default prop/function values | function greet(name = 'Guest') |
| Classes & this | Class components, OOP | class App extends React.Component |
| Modules | Reusable code | import Header from './Header' |
| Promises/Async-Await | API calls, async code | useEffect(() => { fetch... }) |
| Array Methods | Render, transform lists | .map(), .filter() |

**🧠 Library vs Framework vs API – Key Differences**

|  |  |  |
| --- | --- | --- |
| Concept | Definition | Example |
| Library | Pre-written code you call in your app. You stay in control. | React, Lodash |
| Framework | Provides a full structure and calls your code. It controls the flow.  Framework calls the Program | Angular, Django, Spring |
| API | A set of functions and rules for interacting with software.  Program calls the API | REST API, Web API, OpenWeather API |

**📌 Simple Analogy:**

💼 **Library** is like ordering food from a menu – you choose what you want.  
🏗 **Framework** is like a set menu – you follow their structure.  
🧾 **API** is like a waiter – you send a request, get a response.

Day 1: React Fundamentals! 💻✨

**🌟 Day 1: React Fundamentals**

**What You’ll Learn:**

* What is React and why it’s popular
* Setting up your development environment
* Understanding JSX
* Creating components
* Using props and basic state

**💡 1. What is React?**

React is a **JavaScript library** for building user interfaces (Single page web application).  
It helps you create **interactive, reusable components** that manage their own state.

✅ Made by Facebook  
✅ Used in modern apps like Instagram, Netflix, WhatsApp

**🛠 2. Setup Environment**

**Option A: Vite (faster & recommended)**

npm create vite@latest

cd my-app

npm install

npm run dev

**Option B: CRA (Create React App)**

npx create-react-app my-app

cd my-app

npm start

You'll need **Node.js** installed before running the above commands.

**💡 3. JSX - JavaScript + XML**

JSX lets you write HTML inside JavaScript:

const element = <h1>Hello, world!</h1>;

✅ Use **camelCase** for HTML attributes (className, onClick)  
✅ Return **only one root element** from a component

**💡 4. Components**

React apps are built using components.

**Functional Component:**

function Welcome() {

return <h2>Welcome to React!</h2>;

}

**Props Example:**

function Greet(props) {

return <h3>Hello, {props.name}!</h3>;

}

<Greet name="Darwin" />

**🔄 5. Basic State with useState**

import { useState } from 'react';

function Counter() {

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

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Add</button>

</div>

);

}

**💡 Practice Task**

✅ Create a small **Profile Card App** with:

* A functional component for a profile
* Props for name, title, and image
* A button to like the profile using useState

**📚 Tools Used**

* **Vite or CRA** for project setup
* **VS Code** as the code editor
* **Chrome DevTools** for debugging