Introduction to Front-End Frameworks

1. What are Front-End Frameworks?

A front-end framework is a **predefined set of tools, libraries, and best practices** that help developers build user interfaces efficiently. These frameworks provide:

- **Reusable components** (buttons, forms, modals, etc.)
- **Predefined styles and layouts**
- **Built-in functionality** for interactivity and state management

Why Use a Front-End Framework?

✅ Faster development
✅ Consistent UI/UX
✅ Better performance and maintainability
✅ Handles complex state management

2. Popular Front-End Frameworks

1. React.js (Library, not a Full Framework)

- Developed by **Facebook (Meta)**
- Uses a **component-based architecture**
- Utilizes a **Virtual DOM** for better performance
- Strong community support

📌 **Use Case:** Best for building **interactive and dynamic** single-page applications (SPAs).

2. Angular (Full Framework)

- Developed by **Google**
- Uses **TypeScript** instead of JavaScript
- Follows **MVC (Model-View-Controller) architecture**
- Provides built-in **dependency injection**

📌 **Use Case:** Suitable for **enterprise-level** applications with complex logic.

3. Vue.js (Progressive Framework)

- Created by **Evan You**
- Lightweight and easy to integrate
- Supports both **declarative and component-based UI**
- Good for **gradual adoption** in existing projects

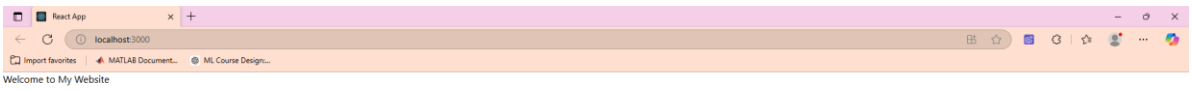
📌 **Use Case:** Best for **small to medium-sized applications** requiring flexibility.

3. Key Concepts in Front-End Frameworks

A. Component-Based Architecture

- The UI is broken into **reusable components**.
- Example: A **Navbar component** can be used across multiple pages.

```
// React Component Example
function Navbar() {
  return <nav>Welcome to My Website</nav>;
}
export default Navbar;
```



B. State Management

- **State** is the data that drives UI updates.
- Managed by internal tools like **React State, Redux, Vuex, NgRx**.

```
// React State Example
const [count, setCount] = useState(0);
```

C. Routing

- Front-end frameworks use **client-side routing** to handle navigation.
- Example: React uses **React Router**, Angular has **Angular Router**.

```
// React Router Example
import { BrowserRouter, Route } from "react-router-dom";
<BrowserRouter>
 <Route path="/home" component={HomePage} />
</BrowserRouter>;
```

4. Advantages and Disadvantages of Front-End Frameworks

Pros	Cons
Speeds up development	Learning curve can be steep
Optimized for performance	Larger bundle size
Ensures maintainability	Frequent updates require adaptation
Cross-browser compatibility	Some frameworks are opinionated

5. How to Choose the Right Framework?

- If you want a **flexible and lightweight** `framework → Vue.js`
- If you are building a **large-scale, enterprise** `project → Angular`
- If you need a **component-based UI with strong community support** `→ React.js`

6. Setting Up a Front-End Framework

Installing React.js

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

Installing Vue.js

```
npm install -g @vue/cli
vue create my-app
cd my-app
npm run serve
```

Installing Angular

```
npm install -g @angular/cli
ng new my-app
cd my-app
ng serve
```

Basics of React.js

1. Introduction to React.js

What is React?

React.js is a **JavaScript library** for building user interfaces. It is:

- **Component-based** (UI is broken into reusable components)
- **Declarative** (describes how the UI should look)
- **Efficient** (uses a Virtual DOM for fast updates)
- **Maintained by Facebook (Meta)** and widely adopted

Why Use React?

- ✅ Faster UI updates with **Virtual DOM**
- ✅ **Reusable components** for maintainability
- ✅ Supports **single-page applications (SPA)**
- ✅ **Strong community support** and vast ecosystem

2. Setting Up a React Project

A. Folder Structure Overview

```
my-app/
├── src/              # Main codebase
│   ├── App.js        # Root component
│   ├── index.js      # Entry point
│   ├── components/   # Custom components
├── public/           # Static assets
├── package.json      # Project dependencies
```

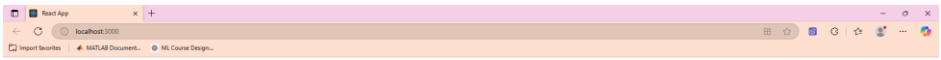
3. Understanding React Components

React is built using **components**, which can be **functional** or **class-based**.

A. Functional Component (Modern Approach)

Uses **functions and hooks** to manage state.

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



B. Class Component (Older Approach)

Uses ES6 classes and the `render()` method.

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

📌 **Functional components are preferred** due to better performance and simplicity.

4. JSX: JavaScript XML

JSX is a syntax extension that allows writing HTML inside JavaScript.

Example of JSX

```
const element = <h1>Welcome to React!</h1>;
```

✅ **Advantages of JSX**:

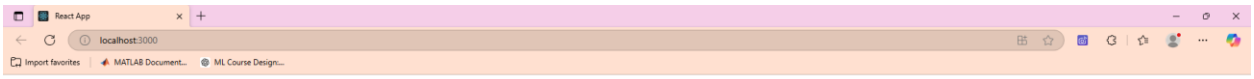
- Makes UI code more readable
- Prevents cross-site scripting (XSS) attacks
- Compiled into `React.createElement()` for performance

5. Props: Passing Data Between Components

Props (**short for properties**) allow components to receive data.

Example: Passing Props

```
function Welcome(props) {
  return <h1>Hello, {props.name}!</h1>;
}
export default function App() {
  return <Welcome name="Goofy" />;
}
```



Hello, Goofy!

📌 **Props are read-only**, meaning they cannot be modified inside the component.

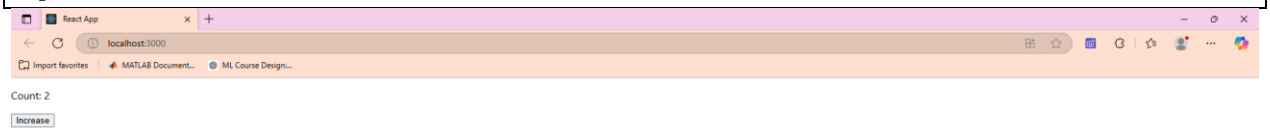
6. State: Managing Component Data

State allows components to manage **dynamic data**.

Using `useState` Hook (Functional Component)

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

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

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



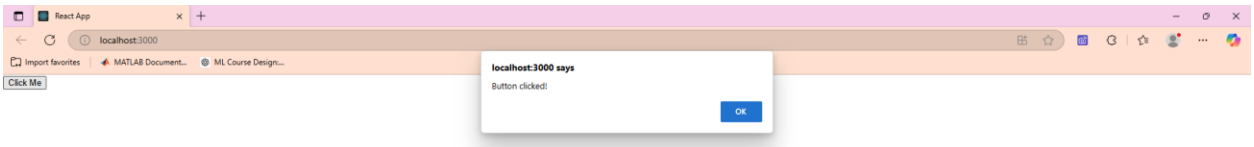
📌 **State is mutable**, unlike props.

7. Handling Events in React

React uses synthetic events, similar to vanilla JS.

Example: Handling Click Events

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



📌 **Event handlers are written in camelCase (`onClick`, `onChange`).**

8. React Lifecycle Methods (Class Components)

For **class components**, React provides lifecycle methods:

Phase	Method	Description
Mounting	`componentDidMount()`	Runs after the component is added to the DOM
Updating	`componentDidUpdate()`	Runs after state or props change
Unmounting	`componentWillUnmount()`	Runs before the component is removed

9. React Hooks (For Functional Components)

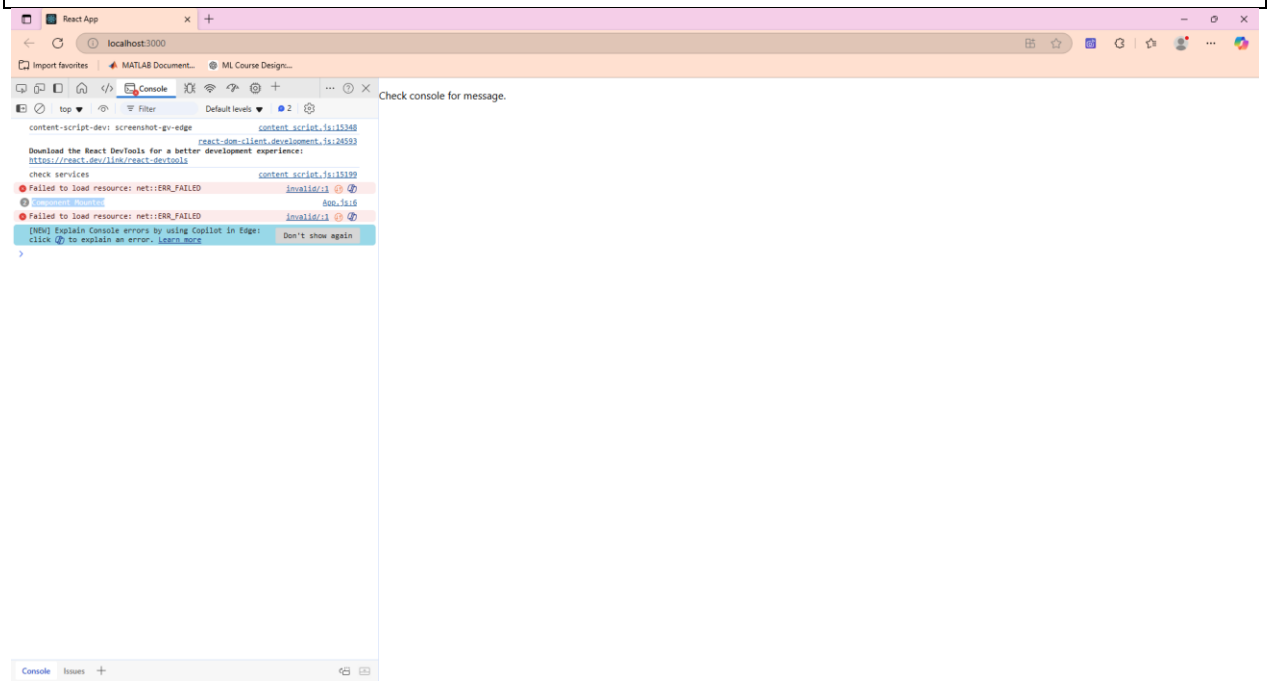
Hooks allow **state and lifecycle features** in functional components.

Hook	Purpose
`useState`	Manages component state

`useEffect`	Handles side effects (API calls, subscriptions)
`useContext`	Shares state between components

Example: `useEffect` Hook

```
import { useEffect } from "react";

function Example() {
 useEffect(() => {
  console.log("Component Mounted");
 }, []); // Empty array = runs only on mount

 return <p>Check console for message.</p>;
}
```



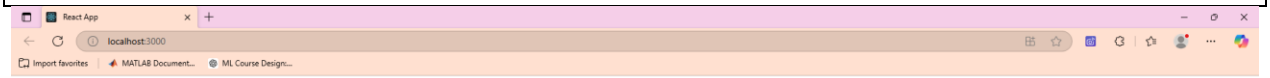
📌 **Replaces `componentDidMount()` in class components.**

10. Conditional Rendering

React allows conditionally displaying components.

Example: If-Else with JSX

```
function Greeting(props) {
 return props.isLoggedIn ? <h1>Welcome Back!</h1> : <h1>Please Log In</h1>;
}
export default function App(){
 return <Greeting isLoggedIn={false}/>;
}
```

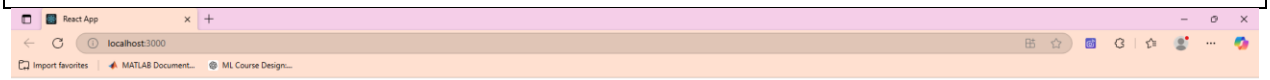


Please Log In

✅ **Use cases:** Showing/hiding UI elements dynamically.

```
import flower from "./flower.jpg";
import './App.css';
const user = {
  name: 'Gaillardia',
  imageSize: 90,
 };

 export default function Profile() {
  return (
   <>
    <h1>{user.name}</h1>
    <img
     className="avatar"
     src={flower}
     alt={'Photo of ' + user.name}
     style={{
```

```
        width: user.imageSize,
        height: user.imageSize
      }}
    />
  </>
 );
}
```

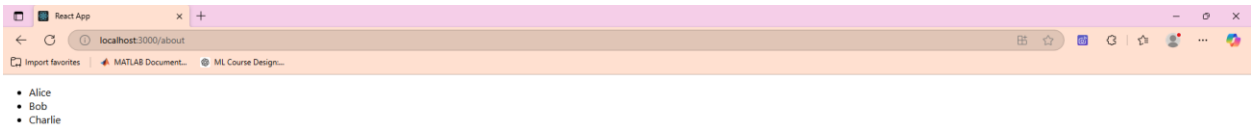


Gaillardia

11. Lists and Keys in React

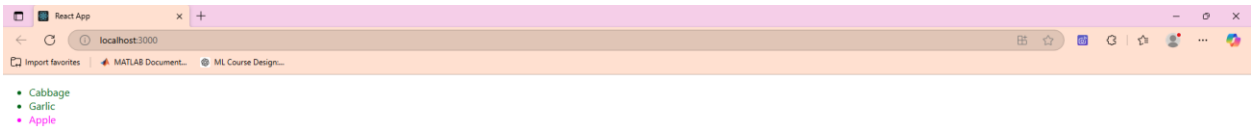
Rendering Lists

```
const users = ["Alice", "Bob", "Charlie"];
return (
 <ul>
  {users.map((user, index) => (
   <li key={index}>{user}</li>
  ))}
 </ul>
);
```



📌 **Keys help React track changes efficiently**.

```
const products = [
  { title: 'Cabbage', isFruit: false, id: 1 },
  { title: 'Garlic', isFruit: false, id: 2 },
  { title: 'Apple', isFruit: true, id: 3 },
 ];
 export default function ShoppingList() {
  const listItems = products.map(product =>
   <li
    key={product.id}
    style={{
     color: product.isFruit ? 'magenta' : 'darkgreen'
    }}
   >
    {product.title}
   </li>
  );
  return (
   <ul>{listItems}</ul>
  );
 }
```



12. React Router (Navigation)

For **single-page applications (SPAs)**, React Router helps manage navigation.

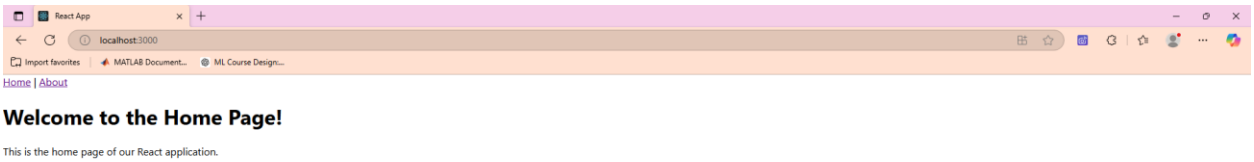
Example: React Router

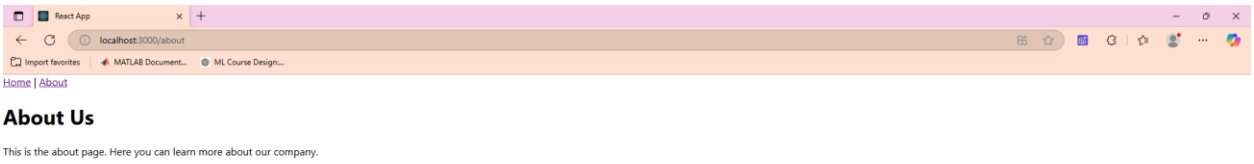
```
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";
import Home from "./components/Home";
import About from "./components/About";

function App() {
 return (
  <BrowserRouter>
   <nav>
    <Link to="/">Home</Link> | <Link to="/about">About</Link>
   </nav>
   <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
   </Routes>
  </BrowserRouter>
 );
}
```

```
export default App;
```

```
import React from "react";

function Home() {
 return (
  <div>
   <h1>Welcome to the Home Page!</h1>
   <p>This is the home page of our React application.</p>
  </div>
 );
}

export default Home;
```

```
import React from "react";

function About() {
 return (
  <div>
   <h1>About Us</h1>
   <p>This is the about page. Here you can learn more about our company.</p>
  </div>
 );
}

export default About;
```





📌 **Routes change without refreshing the page**.

Components, Props, and State

1. Introduction to React Components

In React, the **UI is built using components**. Components:

- Are **reusable**, self-contained pieces of UI.
- Allow **modular** application development.
- Can be **functional or class-based**.

Types of Components

Type	Description
Functional Components	Modern, simpler, and use **React Hooks**.
Class Components	Older, use **ES6 classes** and lifecycle methods.

2. Props with Objects

Props can also **pass objects**.

Example: Passing an Object as a Prop

```
const user = { name: "Gargi", age: 25 };

function UserDetails(props) {

  return <h1>{props.data.name} is {props.data.age} years old.</h1>;

}

export default function App() {

  return <UserDetails data={user} />;

}
```

D. Props with Arrays

Props can pass **lists of data**.

Example: Passing an Array as a Prop

```
const fruits = ["Apple", "Banana", "Cherry"];

function FruitList(props) {
```

```
 return (
  <ul>
   {props.items.map((fruit, index) => (
    <li key={index}>{fruit}</li>
   ))}
  </ul>
 );
}
export default function App() {
 return <FruitList items={fruits} />;
}
```

📌 **Each list item needs a `key` to optimize performance.**

4. State in React

A. What is State?

State is **mutable data** that controls component behavior.

- Unlike **props (which are immutable)**, **state can change**.
- When state changes, **the component re-renders**.

5. Props vs. State: Key Differences

Feature	Props	State
Mutability	**Immutable** (cannot be changed)	**Mutable** (can change over time)
Who Controls It?	**Parent component**	**Component itself**

Can Be Updated?	❌ No	✅ Yes (`useState`, `setState`)
Re-Renders?	✅ Yes (when parent updates)	✅ Yes (when changed)
Example Usage	Passing data between components	Managing counter, form inputs

6. Conditional Rendering with State

State is useful for **showing/hiding UI elements**.

Example: Conditional Rendering

```
import { useState } from "react";
export default function ToggleText() {
  const [isVisible, setIsVisible] = useState(true);
  return (
    <div>
     {isVisible && <p>This is visible</p>}
     <button onClick={() => setIsVisible(!isVisible)}>Toggle</button>
    </div>
  );
 }
```

📌 **&& operator** is used to conditionally render elements.

7. Lifting State Up

When multiple components **share the same state**, move state to a **common parent component**.

Example: Lifting State Up

```
import { useState } from "react";
function Parent() {
 const [message, setMessage] = useState("Hello");

 return (
  <div>
   <Child message={message} />
   <button onClick={() => setMessage("Updated!")}>Change Message</button>
  </div>
```

```
 );
}

function Child(props) {
 return <h1>{props.message}</h1>;
}
```

📌 **The state is lifted to the `Parent` component** and passed to `Child` as a prop.

React Router for Navigation

1. Introduction to React Router

React Router is a **routing library for React** that enables navigation between different views or components in a single-page application (SPA) **without requiring a full page reload**.

Why Use React Router?

- ✅ **Enables client-side routing**
- ✅ **Improves performance** (no full page reloads)
- ✅ **Supports dynamic routing**
- ✅ **Handles nested routes and route parameters**
- ✅ **Provides navigation control (history, redirects, etc.)**

2. Installing React Router

To use React Router, install it using npm or yarn:

```
npm install react-router-dom
```

or

```
yarn add react-router-dom
```

3. Setting Up Basic Routing

React Router provides different components for navigation:

Component	Description
`BrowserRouter`	Wraps the app and enables routing
`Routes`	Contains all route definitions
`Route`	Defines individual routes
`Link`	Used for navigation without reloading the page
`NavLink`	Similar to `Link` but allows active styling
`useNavigate`	Programmatically navigate between pages

4. Using `NavLink` for Active Styling

The `<NavLink>` component allows styling the active link.

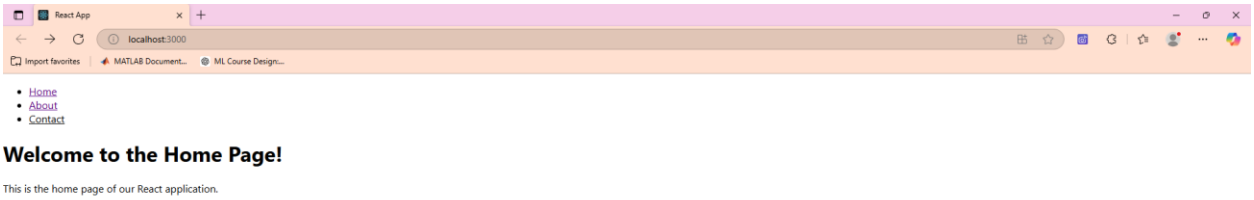
Example: Navigation with Active Link

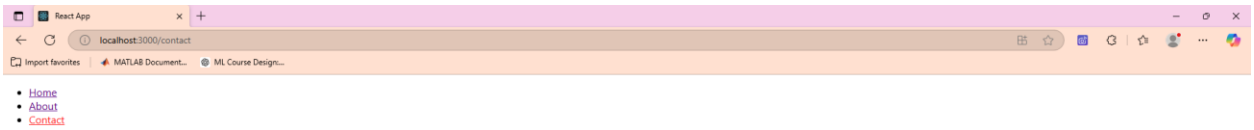
```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Navbar from "./Navbar";
import Home from "./components/Home";
import About from "./components/About";
function App() {
 return (
  <BrowserRouter>
   <Navbar />
   <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />

   </Routes>
  </BrowserRouter>
```

```
 );
}
export default App;
```

```
import { NavLink } from "react-router-dom";
function Navbar() {
 return (
  <nav>
   <ul>
    <li><NavLink to="/" >Home</NavLink></li>
    <li><NavLink to="/about">About</NavLink></li>
    <li><NavLink to="/contact" style={({ isActive }) => ({ color: isActive ? "red" : "black"
})}>Contact</NavLink></li>
   </ul>
  </nav>
 );
}
export default Navbar;
```

📌 **`isActive` is a special property that lets you change styles based on active route.**





localhost:3000/contact

6. Dynamic Routing with URL Parameters

You can pass **dynamic parameters** using `:param` in the route path.

Example: Using Route Parameters

```
import { useParams } from "react-router-dom";

function UserProfile() {
 let { username } = useParams();
 return <h1>Welcome, {username}!</h1>;
}
```

📌 **Define Dynamic Route in `App.js`**

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import UserProfile from "./components/UserProfile";
export default function App() {
 return (
  <BrowserRouter>
   <Navbar />
   <Routes>
        <Route path="/user/:username" element={<UserProfile />} />
   </Routes>
```

```
  </BrowserRouter>
 );
}
```

📌 **Navigate to:**

```
http://localhost:3000/user/Garfield!
```

✓ **It will display:** `Welcome, Garfield!`