Frontend Frameworks — React

Web Technologies — Lecture 8

Masoud Hamad

State University of Zanzibar (SUZA)

Semester II, 2025/2026

Outline

The Problem with Vanilla JS

Building a non-trivial UI with raw DOM manipulation leads to:

- Tangled state spread across the DOM
- Manual sync between data and view
- Hard-to-test code
- Brittle when requirements change

What a framework gives you

Component model, declarative rendering, predictable state management, ecosystem of reusable libraries, tooling.

The Landscape

- **React** (Meta, 2013) — most popular, huge ecosystem
- **Vue** (2014) — approachable, progressive adoption
- **Angular** (Google, 2016) — batteries-included, opinionated
- **Svelte** — compiles to plain JS, very small bundles
- **SolidJS, Qwik** — newer, performance-focused

We'll focus on **React** — biggest job market and most transferable concepts.

Project Setup — Vite

```
npm create vite@latest my-app -- --template react
cd my-app
npm install
npm run dev
```

Vite gives you a fast dev server with hot reload.

A Component is a Function

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

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

JSX is JavaScript with HTML-like syntax. It compiles to function calls.

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)}>+1</button>
      <button onClick={() => setCount(0)}>Reset</button>
    </div>
  );
}
```

Side Effects with useEffect

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

function Students() {
  const [list, setList] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch("/api/students")
      .then(r => r.json())
      .then(data => { setList(data); setLoading(false); });
  }, []);  // [] = run once on mount

  if (loading) return <p>Loading...</p>;
  return <ul>{list.map(s => <li key={s.id}>{s.name}</li>)}</ul>;
}
```

Forms (Controlled Inputs)

```
function NewStudent({ onAdd }) {
  const [name, setName] = useState("");

  function handleSubmit(e) {
    e.preventDefault();
    onAdd({ name });
    setName("");
  }

  return (
    <form onSubmit={handleSubmit}>
      <input value={name} onChange={e => setName(e.target.value)} />
      <button type="submit">Add</button>
    </form>
  );
}
```

Lifting State Up

```
function App() {
  const [students, setStudents] = useState([]);

  function addStudent(s) {
    setStudents([...students, { id: Date.now(), ...s }]);
  }

  return (
    <>
      <NewStudent onAdd={addStudent} />
      <StudentList students={students} />
    </>
  );
}
```

Pattern: **state lives at the lowest common ancestor**, child components receive value + callback (same as Compose).

Client-side Routing — React Router

```
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link> | <Link to="/students">Students</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/students" element={<Students />} />
        <Route path="/students/:id" element={<StudentDetail />} />
      </Routes>
    </BrowserRouter>
  );
}
```

Suggested Project Layout

```
src/
  components/          shared UI: Button, Card, Modal
  pages/               one file per route
  hooks/               custom hooks (useAuth, useFetch)
  lib/                 api.js, helpers
  App.jsx
  main.jsx
public/
```

Vue — Quick Look

```
<template>
  <button @click="count++">Count: {{ count }}</button>
</template>

<script setup>
import { ref } from 'vue';
const count = ref(0);
</script>
```

Vue's Single-File Components keep template, script, and style in one `.vue` file. More approachable for beginners; smaller community than React.

Angular & Svelte

- **Angular** — TypeScript-first, dependency injection, RxJS, opinionated. Common in large enterprise apps.
- **Svelte** — "write less code" — compiles your components to vanilla JS at build time. Tiny runtime, fast.

All four frameworks share the same core ideas: **components**, **reactive state**, **declarative rendering**.

Summary

- React = components + state + props + JSX
- `useState` for state, `useEffect` for side effects
- Lift state up; pass values down, events back up
- React Router for client-side navigation
- Vue, Angular, Svelte solve the same problem with different trade-offs

Next: Databases for the web — SQL and ORMs.

Questions?