TECNOLOGIA SETÚBAL
SETÚBAL POLYTECHNIC UNIVERSITY

Advanced React & Ecosystem

Programação Avançada para a Internet

Mestrado em Engenharia de Software

Prerequisites for this Session

To follow along live:

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

→ Open `src/App.jsx` and replace its entire content with the code from each slide.

Best Practices Note

In these examples, we'll often put all the code in `App.jsx` for simplicity. In a real application, it's best practice to break components into separate files for better organization and maintainability.

Part 1: The Hook System

Moving beyond simple state

useEffect Hook

The useEffect Hook lets you perform side effects in function components, such as data fetching, subscriptions, or manually changing the DOM. Synchronize your component with the outside world (API calls, WebSockets, document.title, localStorage, etc.)

```
useEffect(setupFunction, dependencyArray)
```

Official documentation: https://react.dev/reference/react/useEffect

Dependency Array Behaviour

Dependency array	When it runs
(none)	After every render
`[]`	Only once on mount (like the old componentDidMount)
`[a, b]`	On mount + whenever `a` or `b` changes

Try it: Dependency Array in Action

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

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

  // Runs only once on mount
  useEffect(() => {
    console.log("App Mounted");
  }, []);

  // Runs every time count changes
  useEffect(() => {
    document.title = `Count: ${count}`;
    console.log("Title Updated");
  }, [count]);

  return (
    <div style={{ padding: 20 }}>
      <button onClick={() => setCount(c => c + 1)}>
        Count: {count}
      </button>
    </div>
  );
}

export default App;
```

Attention StrictMode

Note about the console in development
You may see a double console logs on the very first load. This is **React 18+** StrictMode doing its job (it intentionally mounts → cleans up → mounts again to help you find bugs).

To see the real production behaviour right now without creating a production build you can:
Open `src/main.jsx` and temporarily comment out `<React.StrictMode>`:

```
// <StrictMode>
  <App />
// </StrictMode>
```

(Remember to uncomment it again later if this is a real project)

`useEffect` Cleanup – Prevent Memory Leaks

Return a function from the effect → React calls it before unmount and before the next run.

Essential for:

- `setInterval` / `setTimeout`
- WebSocket / EventSource connections
- Event listeners
- Subscriptions

Try it: Cleanup Cycle (App.jsx: Part 1)

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

function StatusIndicator() {
  useEffect(() => {
    console.log("Connecting to System...");

    const intervalId = setInterval(() => {
      console.log("... pinging server ...");
    }, 1000);

    return () => {
      clearInterval(intervalId);
      console.log("Disconnected / Cleaned up");
    };
  }, []); // ← runs only on mount & unmount

  return (
    <div style={{ background: '#eef', padding: 12, borderRadius: 8 }}>
      Connection Active
    </div>
  );
}
```

Try it: Cleanup Cycle (App.jsx: Part 2)

```jsx
function App() {
  const [show, setShow] = useState(false);

  return (
    <div style={{ padding: 30, fontFamily: 'system-ui' }}>
      <h2>useEffect Cleanup Demo</h2>

      <button onClick={() => setShow(s => !s)} style={{ padding: '10px 20px', fontSize: 18 }}>
        {show ? 'Hide Status' : 'Show Status'}
      </button>

      <hr style={{ margin: '30px 0' }} />

      {show && <StatusIndicator />}
    </div>
  );
}

export default App;
```

useRef Hook

The useRef Hook returns a mutable ref object whose .current property is initialized to the passed argument. A box that persists across renders but does NOT trigger re-renders.

Two main uses:

1. Direct DOM access
2. Storing mutable values (timer IDs, previous state, etc.)

Official documentation: https://react.dev/reference/react/useRef

Try it: Auto-focus Input

```
import { useEffect, useRef } from 'react';

function App() {
  const inputRef = useRef(null);

  useEffect(() => {
    inputRef.current?.focus();
    inputRef.current.style.border = "3px solid green";
  }, []);

  return (
    <div style={{ padding: 40 }}>
      <h2>Login</h2>
      <input
        ref={inputRef}
        type="text"
        placeholder="Username (auto-focused)"
        style={{ padding: 10, fontSize: 18 }}
      />
    </div>
  );
}

export default App;
```

Try it: Count renders

```
import { useState, useRef, useEffect } from 'react';

function App() {
  const [inputValue, setInputValue] = useState("");
  const count = useRef(0);

  useEffect(() => {
    count.current = count.current + 1;
  });

  return (
    <>
      <p>Type in the input field:</p>
      <input
        type="text"
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
      />
      <h1>Render Count: {count.current}</h1>
    </>
  );
}

export default App;
```

useContext Hook

The useContext Hook lets you subscribe to React context without introducing nesting. Solves prop drilling by providing a global "teleport" for data like theme, auth user, language, shopping cart, etc.

Official documentation: https://react.dev/reference/react/useContext

Problem: Prop Drilling Hell

Imagine you need the current theme (or user, language, etc.) in a deeply nested component:

```
App
 └── Layout
      └── Header
           └── Nav
                └── UserMenu
                     └── Avatar   ← only this one needs the theme!
```

Without Context → you pass `theme` as a prop through **5 levels** that don't need it.

Try it: Global Theme Toggle (App.jsx: Part 1)

```jsx
import { createContext, useContext } from 'react';

// 1. Create the Context (can be exported to its own file later)
const ThemeContext = createContext(null);

// Deep child - no props received!
function ThemedButton() {
  // 3. Consume the context anywhere below the Provider
  const { theme, toggle } = useContext(ThemeContext);

  return (
    <button
      onClick={toggle}
      style={{
        background: theme === 'dark' ? '#333' : '#fff',
        color: theme === 'dark' ? '#fff' : '#000',
        padding: '12px 24px',
        fontSize: 18,
        border: '2px solid #000',
        borderRadius: 8,
      }}
    >
      Theme: {theme.toUpperCase()}
    </button>
  );
}
```

Try it: Global Theme Toggle (App.jsx: Part 2)

```
import { useState } from 'react';
// ... import ThemeContext and ThemedButton from above

function App() {
  const [theme, setTheme] = useState('light');
  const toggle = () => setTheme(t => (t === 'light' ? 'dark' : 'light'));

  return (
    // 2. Provide the value high in the tree
    <ThemeContext.Provider value={{ theme, toggle }}>
      <div style={{
        padding: 60,
        minHeight: '100vh',
        transition: 'background 0.3s',
        background: theme === 'dark' ? '#000' : '#f0f0f0',
        color: theme === 'dark' ? '#fff' : '#000'
      }}>
        <h1>Context API Demo</h1>
        <p>Click the button – the whole page background changes too!</p>
        <ThemedButton />
      </div>
    </ThemeContext.Provider>
  );
}

export default App;
```

useMemo Hook

The useMemo Hook lets you cache the result of an expensive calculation between re-renders. Caches result of calculation for heavy filtering/sorting of large lists.

Official documentation: https://react.dev/reference/react/useMemo

useCallback Hook

The useCallback Hook lets you cache a function definition between re-renders. Caches function definition for stable callbacks in React.memo children.

Official documentation: https://react.dev/reference/react/useCallback

Performance Hooks

Hook	Caches	Typical Use Case
`useMemo`	Result of calculation	Heavy filtering/sorting of large lists
`useCallback`	Function definition	Stable callbacks for `React.memo` children

Only use when you measure a real performance issue.

Try it: useMemo for Expensive Calculation

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

function App() {
  const [count, setCount] = useState(0);
  const [numbers] = useState(Array.from({length: 1000000}, (_, i) => i));

  // Without useMemo: re-filters on every render (slow!)
  // const filtered = numbers.filter(n => n % 2 === 0);

  // With useMemo: only when 'numbers' changes
  const filtered = useMemo(() => {
    console.log('Filtering...'); // Logs only once
    return numbers.filter(n => n % 2 === 0);
  }, [numbers]);

  return (
    <div style={{ padding: 20 }}>
      <button onClick={() => setCount(c => c + 1)}>Re-render: {count}</button>
      <p>Even numbers count: {filtered.length}</p>
    </div>
  );
}

export default App;
```

Try it: useCallback for Stable Functions

```jsx
import { useState, useCallback, memo } from 'react';

// Child component – memoized to prevent re-renders
const Child = memo(({ onClick }) => {
  console.log('Child rendered'); // Should log only once
  return <button onClick={onClick}>Click me</button>;
});

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

  // Without useCallback: new function every render → Child re-renders
  // const handleClick = () => console.log('Clicked');

  // With useCallback: stable function
  const handleClick = useCallback(() => {
    console.log('Clicked');
  }, []);

  return (
    <div style={{ padding: 20 }}>
      <button onClick={() => setCount(c => c + 1)}>Parent: {count}</button>
      <Child onClick={handleClick} />
    </div>
  );
}

export default App;
```

Documentation & Further Reading

- React.dev (The New Official Docs)
- MDN Web Docs: React
- W3Schools React Tutorial
- React Roadmap (roadmap.sh)

Part 2: Professional Ecosystem

Some Extra Libraries for Real Apps

Need	Library/Options	Why
Routing	`react-router-dom`	SPA navigation
Styling	Tailwind CSS Bootstrap DaisyUI (on Tailwind) Sass (SCSS)	Rapid, consistent UI
Drag & Drop	`@dnd-kit`	Modern, accessible, mobile-friendly

react-router-dom

react-router-dom is the most popular routing library for React web apps, enabling client-side routing for SPAs – navigate without full page reloads. Essential for multi-page apps (e.g., /home, /about, /user/:id).

Install: `npm install react-router-dom`

Official documentation: https://reactrouter.com/

Try it: Basic Routing (main.jsx)

Update `src/main.jsx`:

```jsx
import { StrictMode } from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom';
import './index.css'
import App from './App.jsx';

ReactDOM.createRoot(document.getElementById('root')).render(
  <StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </StrictMode>
);
```

Try it: Basic Routing (App.jsx)

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

function Home() { return <h1>Home Page</h1>; }
function About() { return <h1>About Page</h1>; }

function App() {
  return (
    <div style={{ padding: 20 }}>
      <nav>
        <Link to="/">Home</Link> | <Link to="/about">About</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </div>
  );
}

export default App;
```

Tailwind CSS

Tailwind CSS is a utility-first CSS framework for rapidly building custom user interfaces. Build UIs fast with zero config.

1. Install:

```
npm install tailwindcss @tailwindcss/vite
```

2. Update `vite.config.js`:

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import tailwindcss from '@tailwindcss/vite'

// https://vite.dev/config/
export default defineConfig({
  plugins: [react(), tailwindcss()],
})
```

Try it: Tailwind CSS Button

```
function App() {
  return (
    <div className="p-10">
      <button className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded">
        Tailwind Button
      </button>
    </div>
  );
}

export default App;
```

DaisyUI

daisyUI is a plugin for Tailwind CSS that provides a collection of accessible and customizable UI components. Beautiful, themeable components on top of Tailwind.

1. Install:

`npm i -D daisyui`

2. Configure (`src/index.css`):

Add the plugin directive directly in your CSS file:

```
@import "tailwindcss";
@plugin "daisyui";
```

Official documentation: https://daisyui.com/

Try it: DaisyUI Button

```
function App() {
  return (
    <div className="p-10">
      <button className="btn btn-primary">
        DaisyUI Button
      </button>
    </div>
  );
}

export default App;
```

Bootstrap

Bootstrap is a powerful, extensible, and feature-packed frontend toolkit. Pre-built components and grid system – quick prototyping.

1. Install:

```
npm install bootstrap
```

2. Import (`src/main.jsx`):

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

Official documentation: https://getbootstrap.com/

Try it: Bootstrap Button

```
function App() {
  return (
    <div className="container p-5">
      <button className="btn btn-primary">
        Bootstrap Button
      </button>
      <div className="mt-3 alert alert-info">
        Grid system active
      </div>
    </div>
  );
}

export default App;
```

Sass

Sass is a stylesheet language that's compiled to CSS, providing features like variables, nesting, and more. CSS with superpowers: Nesting for cleaner, hierarchical CSS; Modules for built-in math and color; Variables for reusable values.

1. Install:

```
npm install -D sass
```

2. Usage:

Rename any `.css` file to `.scss`. Vite supports it out of the box.

Official documentation: https://sass-lang.com/

Try it: Sass Nesting & Modules

1. Create `src/App.scss`:

```scss
@use "sass:color"; /* Required for modern color manipulation */

$primary-color: #646cff;

.card {
  padding: 2em;
  background: #f9f9f9;
  border-radius: 8px;
  text-align: center;

  h2 {
    color: $primary-color;
    /* Modern syntax replacing deprecated darken() */
    &:hover {
      color: color.adjust($primary-color, $lightness: -20%);
    }
  }
}
```

2. Update `src/App.jsx`:

```jsx
import './App.scss';

function App() {
  return (
    <div className="card">
      <h2>Sass Power</h2>
      <p>This uses modern Sass modules and nesting.</p>
    </div>
  );
}

export default App;
```

@dnd-kit

@dnd-kit is a lightweight, modular, performant, accessible and extensible drag & drop toolkit for React. Modern, accessible, mobile-friendly drag & drop.

Official documentation: https://docs.dndkit.com/

Installation:

```
npm install @dnd-kit/core @dnd-kit/utilities
```

Try it: @dnd-kit Draggable (Desktop + Mobile) – Part 1

```
import {
  DndContext,
  useDraggable,
  useSensor,
  useSensors,
  PointerSensor,
} from '@dnd-kit/core';
import { CSS } from '@dnd-kit/utilities';

function DraggableBox() {
  const { attributes, listeners, setNodeRef, transform } = useDraggable({ id: 'unique-box' });

  const style = {
    transform: CSS.Translate.toString(transform),
    width: 120, height: 120, background: 'royalblue', color: 'white',
    borderRadius: 12, display: 'flex', alignItems: 'center',
    justifyContent: 'center', cursor: 'grab', userSelect: 'none',
    touchAction: 'none', // <-- CRITICAL for mobile to prevent scrolling
  };

  return (
    <div ref={setNodeRef} style={style} {...listeners} {...attributes}>
      Drag Me!
    </div>
  );
}
```

Try it: @dnd-kit Draggable (Desktop + Mobile) – Part 2

```
// ... import from above

function App() {
  // Activation constraint fixes "sticky" feeling on touch devices
  const sensors = useSensors(
    useSensor(PointerSensor, {
      activationConstraint: {
        distance: 8,
      },
    })
  );

  return (
    <DndContext sensors={sensors}>
      <div style={{ padding: 80 }}>
        <h2>@dnd-kit works on desktop & mobile</h2>
        <DraggableBox />
      </div>
    </DndContext>
  );
}

export default App;
```

Thank You!

Questions?