CURS 7
React - Fundamente

7.1 De ce React? Problema pe care o rezolvă

De ce React?
React este biblioteca JavaScript creată de Meta (Facebook) pentru construirea de interfețe de utilizator. Este cea mai folosită bibliotecă front-end din lume (2024): ~40% din proiectele web noi aleg React. Cursurile 7 și 8 acoperă tot ce ai nevoie pentru a construi aplicații React reale, de la componentul simplu până la arhitectura completă cu routing și state management.

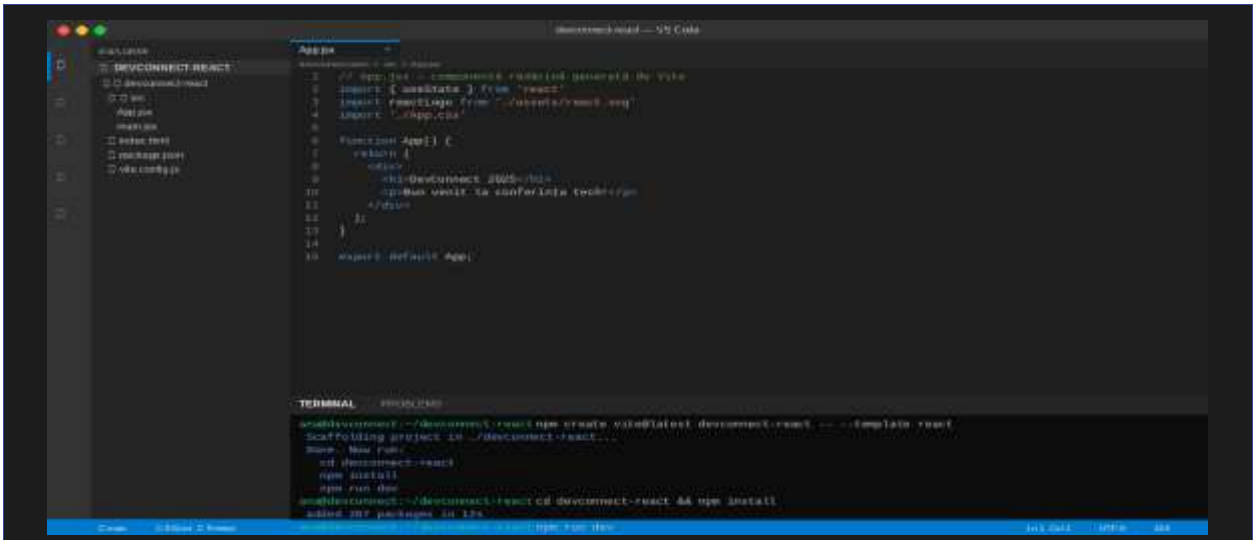
Înainte de biblioteci ca React, interfețele complexe se construiau cu manipulare directă a DOM-ului în JavaScript vanilla. Abordarea funcționa pentru pagini simple, dar devenea imposibil de menținut pentru aplicații cu zeci de componente interactive care trebuie să se sincronizeze.

JavaScript Vanilla (abordarea anterioară)	React (abordarea modernă)
document.querySelector() peste tot	Componente reutilizabile cu props
Sincronizare manuală DOM ↔ date	UI = f(state) - React actualizează DOM automat
Event listeners adăugați manual	onClic, onChange declarativ în JSX
Greu de reutilizat bucăți de UI	Import component oriunde ai nevoie
Fără structură impusă	Arhitectură clară: componente → pagini → app

Principiul fundamental React: UI = f(state). Interfața este o funcție deterministă a stării. Când starea se schimbă, React recalculează ce trebuie să se schimbe în DOM și aplică doar diferențele (reconciliation / Virtual DOM). Tu nu manipulezi DOM-ul direct - React face asta pentru tine.

7.2 Setup proiect cu Vite

Vite este build tool-ul modern pentru proiecte React (înlocuiește Create React App, care nu mai este recomandat). Este de 10-100x mai rapid la development datorită ES Modules native.



VS Code: terminalul arată npm create vite@latest, scaffolding instant, npm install (287 pachete în 12s) și npm run dev pornind serverul de development pe localhost:5173 cu Hot Module Replacement activ.

```
# Crearea proiectului DevConnect cu React + Vite
npm create vite@latest devconnect-react -- --template react
cd devconnect-react
npm install
npm run dev      # pornește pe http://localhost:5173

# Structura generată:
devconnect-react/
├── src/
|   ├── App.jsx          # componenta rădăcină
|   ├── main.jsx         # punct de intrare (ReactDOM.render)
|   └── assets/
├── index.html           # șablonul HTML (un singur <div id='root'>)
├── package.json
└── vite.config.js       # configurare Vite (proxy API etc.)

# Structura recomandată pentru DevConnect:
src/
├── components/          # componente reutilizabile (SpeakerCard, NavBar)
├── pages/               # pagini corespunzătoare rutelor (Home, Login)
├── hooks/               # custom hooks (useSpeakeri, useFormular)
├── context/             # Context API (AuthContext)
├── api/                 # toate fetch()-urile (din Lab 6)
└── utils/               # funcții helper pure
```

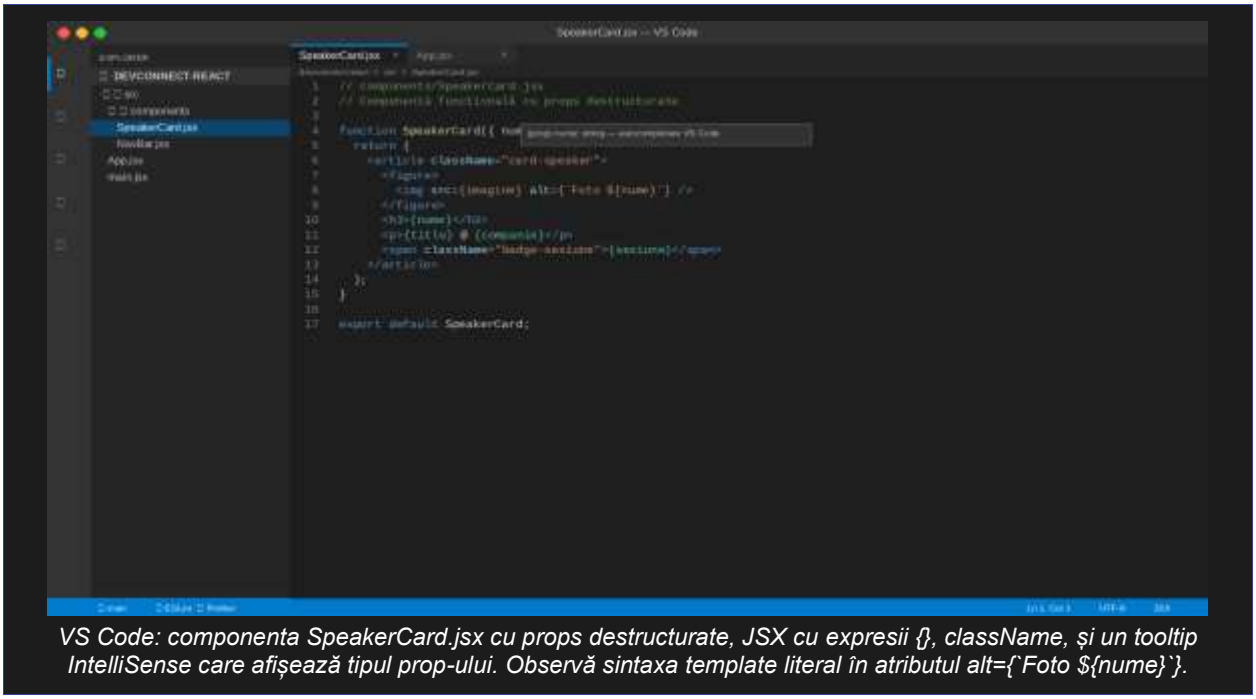
Instalează extensia 'ES7+ React/Redux/React-Native snippets' (dsznajder) pentru shortcut-uri: rafce → generează un React functional component cu export default în 0.5 secunde.

7.3 JSX - JavaScript XML

JSX este o extensie de sintaxă JavaScript care permite scrierea de HTML-like în fișiere .jsx. Nu este HTML real - este transformat de Babel/Vite în apeluri React.createElement(). Cunoașterea diferențelor față de HTML este esențială.

Concept	HTML	JSX (diferență)
Atribut clasă CSS	class="card"	className="card" (class e rezervat în JS)
Atribut for (label)	for="email"	htmlFor="email"

Taguri self-closing	<input> sau 	<input /> sau (obligatoriu /)
Comentarii	<!-- comentariu -->	{/* comentariu */}
Expresii JS	Nu e posibil	{variabila} sau {conditie && }
Stiluri inline	style="color:red"	style={{color:'red'}} (obiect JS)
Event handlers	onclick="fn()"	onClick={fn} (camelCase, funcție, nu string)



VS Code: componenta SpeakerCard.jsx cu props destructurate, JSX cu expresii {}, className, și un tooltip IntelliSense care afișează tipul prop-ului. Observă sintaxa template literal în atributul alt={`Foto ${nume}`}.

Props - comunicarea între componente

Props (prescurtare de la properties) sunt argumentele componentelor React. Se transmit unidirecțional: părintele transmite date copilului, copilul nu modifică props primite (read-only).

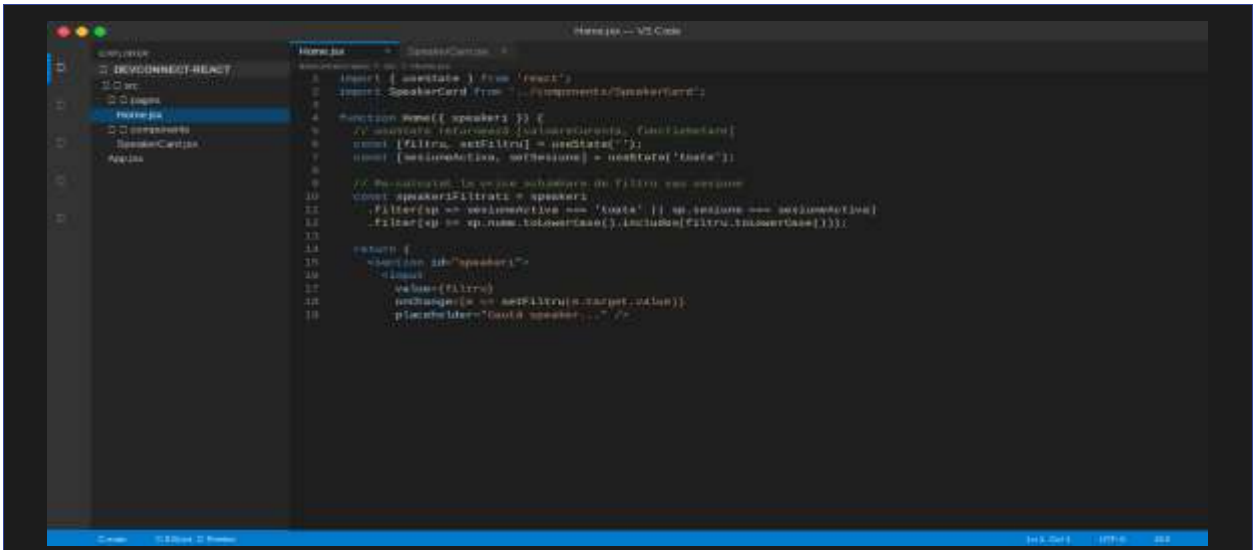
```
// Transmiterea props - componenta părinte (App.jsx)
<SpeakerCard
  nume="Ana Ionescu"
  titlu="CTO @ TechRomania"
  sesiune="Keynote"
  imagine={speakerImgUrl}
  onSelect={handleSelectSpeaker}   // funcție ca prop (callback)
/>

// Primirea props - componenta copil (SpeakerCard.jsx)
// Varianta 1: obiect props
function SpeakerCard(props) {
  return <h3>{props.nume}</h3>;
}

// Varianta 2: destructurare (recomandat)
function SpeakerCard({ nume, titlu, sesiune, imagine, onSelect }) {
  return (
    <article onClick={() => onSelect({ nume, titlu })} className="card">
      <img src={imagine} alt={`Foto ${nume}`} />
      <h3>{nume}</h3>
      <p>{titlu}</p>
```

```
        <span className="badge">{sesiune}</span>
      </article>
    );
  }

  // Props speciale:
  // children - conținutul dintre taguri: <Card><p>text</p></Card>
  // key      - OBLIGATORIU în liste: .map(sp => <Card key={sp.id} />)
```

> ⚠ Prop-ul 'key' în liste este obligatoriu și trebuie să fie unic și stabil (ID din baza de date, nu indexul array-ului). Fără key, React nu poate identifica eficient ce element s-a schimbat și poate produce bug-uri subtile la reordonare sau ștergere.

7.4 useState - State local în componente

useState este cel mai important hook React. Stochează date locale ale componentei și declansează o re-renderizare de fiecare dată când valoarea se schimbă.



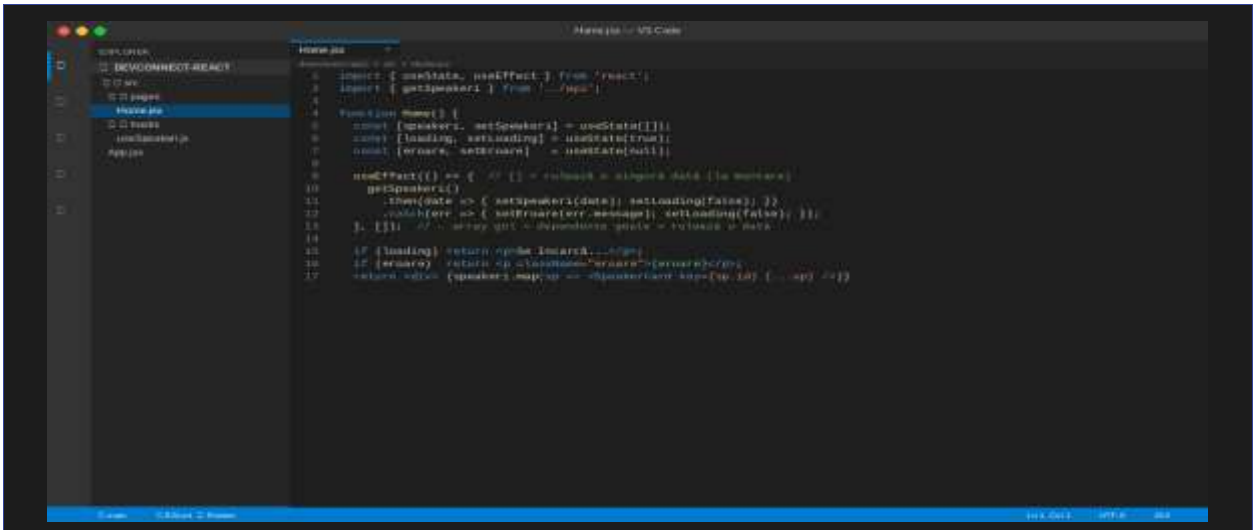
VS Code: pagina Home.jsx cu două useState-uri (filtru string și sesiuneActiva). Observă că speakeriFiltrati este recalculat la fiecare render prin chaining de .filter() - nu necesită state separat deoarece este derivat din state existent.

```
  // Sintaxa useState
  const [valoare, setValoare] = useState(valoareInitiala);
  //     ↑ state curent  ↑ funcție de actualizare  ↑ valoare inițială

  // Exemple din DevConnect:
  const [filtru, setFiltru]         = useState('');
  const [sesiune, setSesiune]       = useState('toate');
  const [loading, setLoading]       = useState(true);
  const [speaker, setSpeaker]       = useState(null);      // obiect/null
  const [workshopsAlese, setWS]     = useState([]);        // array
  const [formData, setFormData]     = useState({           // obiect
    nume: '', email: '', workshop: ''
  });

  // REGULI CRITICE:
  // ✗ NU modifica state direct - React nu detectează schimbarea:
  speakeri.push(nouSpeaker);  // GREȘIT - nu declanșează re-render

  // ☑ Folosește setter cu valoare nouă:
  setSpeakeri([...speakeri, nouSpeaker]);  // spread + element nou
  setSpeakeri(prev => [...prev, nouSpeaker]);  // cu funcție (mai sigur)
```

```
// ☑ Actualizare obiect - spread pentru a păstra câmpurile existente:
setFormData(prev => ({ ...prev, email: 'nou@email.ro' }));
```

7.5 useEffect - Efecte secundare

useEffect rulează cod după ce componenta s-a randat. Este mecanismul pentru: fetch de date, subscripții la evenimente, timere, integrare cu librării third-party - orice operație care trebuie să se întâmple 'în afara' randării.



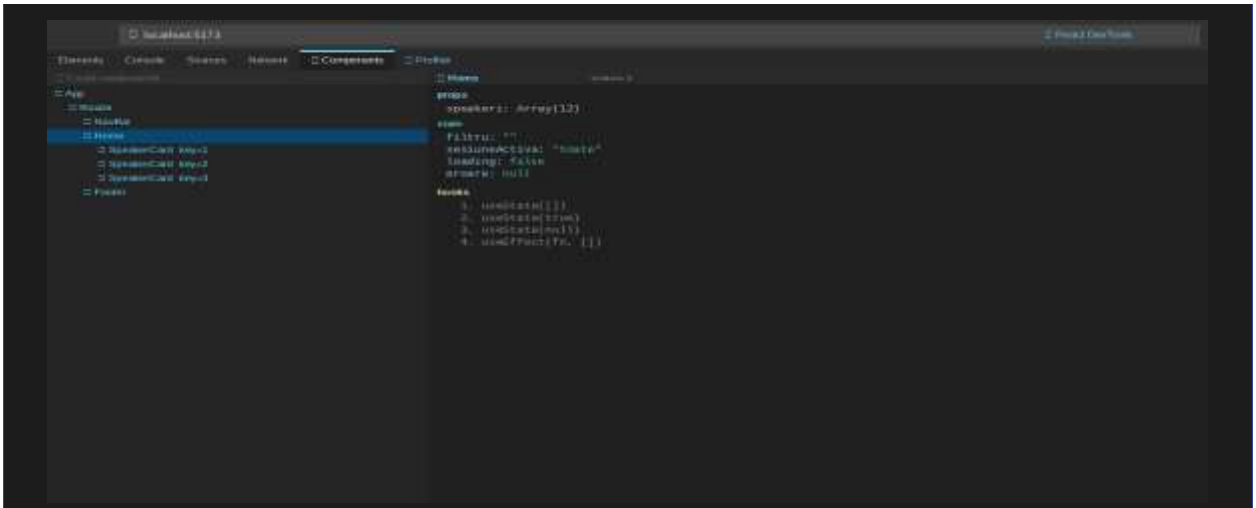
VS Code: Home.jsx cu useEffect pentru fetch date la montare, gestionând toate cele trei stări: loading (true la start), eroare (null/string), speakeri (array). Array-ul gol [] ca al doilea argument garantează că efectul rulează o singură dată.

Al doilea argument	Când rulează useEffect
useEffect(fn)	La fiecare render (rar util, poate cauza bucle infinite)
useEffect(fn, [])	O singură dată, la montarea componentei (fetch inițial)
useEffect(fn, [id])	La montare ȘI la fiecare schimbare a variabilei id
useEffect(fn, [a, b])	La montare ȘI când a sau b se schimbă
return () => cleanup()	Funcția returnată rulează la demontare (cleanup timere, subscripții)

```
// Pattern standard: fetch cu cleanup (previne memory leaks)
useEffect(() => {
  let activ = true;  // flag pentru a preveni actualizare după demontare

  async function incarca() {
    try {
      setLoading(true);
      const date = await getSpeakeri();
      if (activ) setSpeakeri(date);  // nu actualizează dacă componenta s-a
demontat
    } catch (err) {
      if (activ) setEroare(err.message);
    } finally {
      if (activ) setLoading(false);
    }
```

```
    }

    incarca();
    return () => { activ = false; };  // cleanup la demontare
  }, []);
```

7.6 React Developer Tools



Browser cu React DevTools: panoul Components arată arborele componentelor (App → Router → NavBar, Home → SpeakerCard×3), iar panoul drept afișează props, state și hooks ale componentei Home selectate. Esențial pentru debugging React.

Instalează extensia React Developer Tools pentru Chrome sau Firefox (developer.chrome.com/docs/devtools). În development, React afișează avertismente detaliate în Console - citește-le întotdeauna, sunt mai clare decât mesajele de eroare obișnuite.

7.8 DevConnect - Exemplificare Completă React Fundamente

Această secțiune arată concret cum migrăm fiecare parte a DevConnect din JavaScript vanilla (Lab 1–6) în componente React. Urmărim același cod pas cu pas: de la HTML static → JS vanilla → componentă React.

Transformarea SpeakerCard: HTML → JS Vanilla → React

Comparație directă între cele trei abordări pentru a înțelege ce câștigăm cu React:

```
// ─── Abordarea 1: HTML static (Lab 1) ──────────────────────────
// Dezavantaj: date hardcodate, nu reutilizabil
<article class="card-speaker">
  <img src="img/speakers/ana-ionescu.jpg" alt="Foto Ana Ionescu">
  <div class="card-body">
    <h3>Ana Ionescu</h3>
    <p class="titlu">CTO @ TechRo</p>
  </div>
</article>

// ─── Abordarea 2: JS Vanilla (Lab 2-6) ─────────────────────────
// Mai bine, dar: DOM API verbos, nici un IntelliSense pe structură
function creeazaCard(sp) {
  const article = document.createElement('article');
  article.className = 'card-speaker';
  article.innerHTML = `
    <img src='${sp.avatar}' alt='Foto ${sp.nume}'>
```

```
      <div class='card-body'>
        <h3>${sp.nume}</h3>
        <p class='titlu'>${sp.titlu}</p>
      </div>
    `;
    return article;
  }

  // ─── Abordarea 3: React (Lab 7) ──────────────────────────────
  // Clar, reutilizabil, TypeScript-friendly, testabil
  function SpeakerCard({ id, nume, titlu, companie, sesiune, avatar, bio, linkedin
  }) {
    const [expandat, setExpandat] = useState(false);

    return (
      <article className='card-speaker'>
        <img
          src={avatar || '/img/speakers/default.jpg'}
          alt={`Foto ${nume}`}
          onError={e => { e.target.src = '/img/speakers/default.jpg'; }}
          loading='lazy'
        />
        <div className='card-body'>
          <h3>{nume}</h3>
          <p className='titlu'>{titlu} @ {companie}</p>
          <span className='badge-sesiune'>{sesiune}</span>
          {expandat && <p className='bio'>{bio}</p>}
          <div className='card-actiuni'>
            <button onClick={() => setExpandat(e => !e)} className='btn-bio'>
              {expandat ? 'Restrânge ▲' : 'Bio complet ▼'}
            </button>
            <a href={linkedin} target='_blank' rel='noopener noreferrer'
               className='btn-linkedin'>LinkedIn →</a>
          </div>
        </div>
      </article>
    );
  }

  export default SpeakerCard;
```

> Câștigurile cu React față de vanilla: (1) Stare locală (expandat) gestionată automat - nu mai căutăm elementul în DOM la click. (2) onError pe img - fallback declarativ, nu addEventListener separat. (3) Condițional {expandat && <p>} - clar, fără toggleClass sau innerHTML manual. (4) Reutilizare trivială: <SpeakerCard {...speaker} /> oriunde.

useEffect pentru încărcarea speakerilor - comparație

Aceeași operație: fetch speakeri + stări loading/error. Vanilla JS din Lab 5 vs useEffect în React:

```
  // ─── Lab 5 - Vanilla JS ──────────────────────────────────────
  // Trebuie să găsim manual elementul DOM la fiecare actualizare
  export async function incarcaSpeakeri() {
    const grid = document.querySelector('#speakeri .grid-speakeri');
    grid.innerHTML = `<div class='spinner'></div>`; // loading manual
    try {
      const speakeri = await getSpeakeri();
      grid.innerHTML = '';                          // curăță manual
      speakeri.forEach(sp => grid.appendChild(creeazaCard(sp)));
    } catch (err) {
      grid.innerHTML = `<p class='eroare'>${err.message}</p>`;
    }
  }
```

```
// ── Lab 7 - React cu useEffect ──────────────────────────────
// UI este o funcție de state - React actualizează DOM automat
function Speakeri() {
  const [speakeri, setSpeakeri] = useState([]);
  const [loading, setLoading]   = useState(true);
  const [eroare, setEroare]     = useState(null);

  useEffect(() => {
    let activ = true; // previne update după demontare (memory leak)
    getSpeakeri()
      .then(data => { if (activ) setSpeakeri(data); })
      .catch(err  => { if (activ) setEroare(err.message); })
      .finally(()  => { if (activ) setLoading(false); });
    return () => { activ = false; }; // cleanup
  }, []); // [] = rulează o singură dată la montare

  // UI = f(state) - React decide ce se schimbă în DOM
  if (loading) return <LoadingSpinner />;
  if (eroare)  return <ErrorMessage mesaj={eroare} />;
  return (
    <div className='grid-speakeri'>
      {speakeri.map(sp => <SpeakerCard key={sp.id} {...sp} />)}
    </div>
  );
}
```

Filtrarea reactivă a speakerilor - fără nicio manipulare DOM

Unul din cele mai ilustrative exemple pentru puterea React: filtrare live a unei liste. În vanilla JS, trebuia să selectăm elementele, să le ascundem/afișăm manual. În React, starea conduce totul:

```
// pages/Speakeri.jsx - filtrare reactivă completă
const SESIUNI = ['toate', 'Keynote', 'Workshop', 'Lightning Talk', 'Panel'];

function Speakeri() {
  const [speakeri, setSpeakeri] = useState([]);
  const [loading, setLoading]   = useState(true);
  const [eroare, setEroare]     = useState(null);
  const [cautare, setCautare]   = useState('');
  const [sesiune, setSesiune]   = useState('toate');

  useEffect(() => {
    getSpeakeri().then(setSpeakeri)
      .catch(e => setEroare(e.message))
      .finally(() => setLoading(false));
  }, []);

  // ── State derivat - calculat la fiecare render, NU useState ──
  // Nu avem nevoie de setSpeakeriFiltrati - e derivat din
speakeri+cautare+sesiune
  const afisati = speakeri
    .filter(sp => sesiune === 'toate' || sp.sesiune === sesiune)
    .filter(sp => [sp.nume, sp.titlu, sp.companie]
      .join(' ').toLowerCase().includes(cautare.toLowerCase()));

  if (loading) return <LoadingSpinner mesaj='Se încarcă speakerii...' />;
  if (eroare)  return (
    <ErrorMessage mesaj={eroare}
      onRetry={() => { setEroare(null); setLoading(true); /* re-fetch */ }}
    />
  );

  return (
    <section id='speakeri'>
      <h2>Speakeri DevConnect 2026</h2>
```

```
        {/* ── Filtre ── */}
        <div className='bara-filtre'>
          <input
            type='search'
            placeholder='Caută după nume, titlu, companie...'
            value={cautare}
            onChange={e => setCautare(e.target.value)} {/* reactiv: la fiecare
  tastă */}
          />
          <div className='filtre-sesiuni' role='group' aria-label='Filtrare
  sesiune'>
            {SESIUNI.map(s => (
              <button key={s}
                className={`btn-filtru ${sesiune === s ? 'activ' : ''}`}
                onClick={() => setSesiune(s)}
                aria-pressed={sesiune === s}
              >
                {s}
              </button>
            ))}
          </div>
        </div>

        {/* ── Rezultate ── */}
        <p className='numar-rezultate'>
          {afisati.length} speaker{afisati.length !== 1 ? 'i' : ''} găsiți
        </p>

        <div className='grid-speakeri'>
          {afisati.map(sp => <SpeakerCard key={sp.id} {...sp} />)}
        </div>

        {afisati.length === 0 && (
          <div className='no-results'>
            <p>Niciun speaker nu corespunde criteriilor.</p>
            <button onClick={() => { setCautare(''); setSesiune('toate'); }}>
              Resetează filtrele
            </button>
          </div>
        )}
      </section>
    );
  }
```

7.9 Referințe Curs 7

- React Docs oficiale - react.dev/learn (cel mai bun punct de start)
- Vite - documentație oficială
- MDN - JavaScript modules (necesar pentru import/export)
- React DevTools - extensie Chrome
- Scrimba - Learn React for Free (interactiv)

Notă privind elaborarea materialelor de curs

Vreau să fiu transparent cu voi: structura și conținutul acestor note de curs au fost generate cu ajutorul unui instrument de inteligență artificială (Claude, de la Anthropic), pe baza cerințelor și direcțiilor pe care le-am formulat eu ca titular de curs.

De ce vă spun asta? Pentru că:
• Nu pot garanta că fiecare noțiune tehnică are 100% acuratete sau este actualizată
• Vă încurajez să verificați activ sursele bibliografice indicate
• Utilizarea responsabilă a AI în educație înseamnă transparență, nu ascundere

Considerați aceste materiale un ghid structurat de studiu, nu un manual definitiv. Dacă identificați o eroare sau o neclaritate, veniți cu ea la curs.