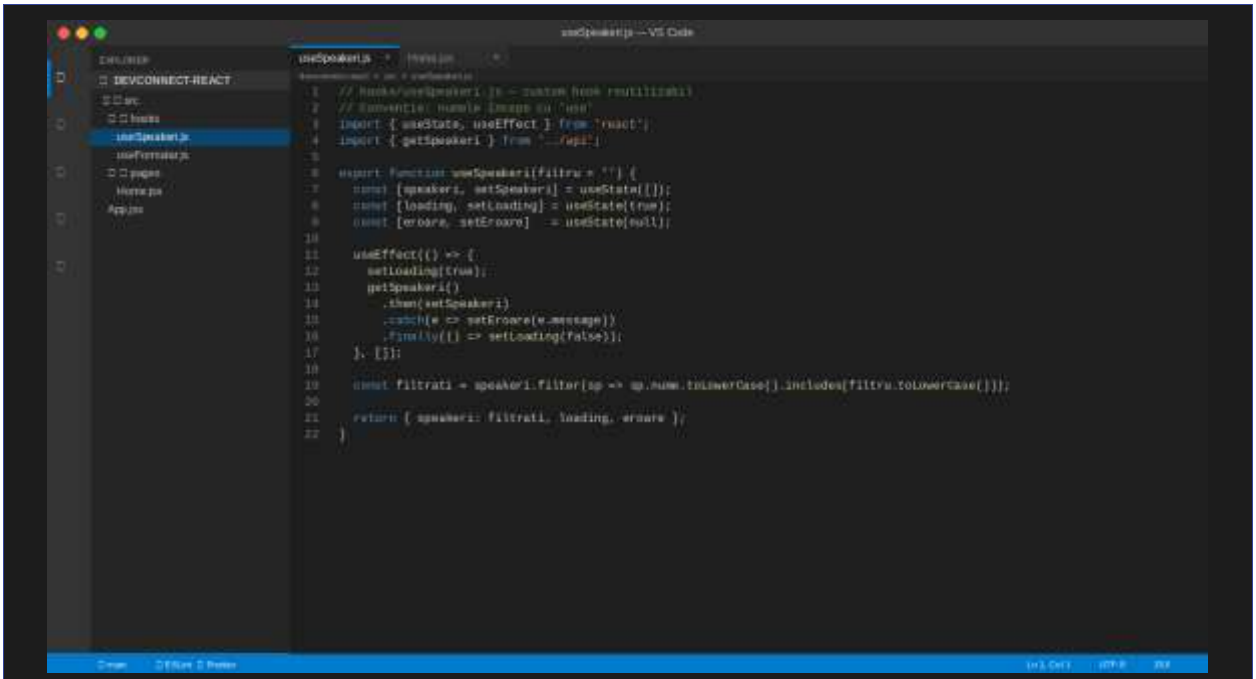
CURS 8

React Avansat

8.1 Custom Hooks - Logică reutilizabilă

Un custom hook este o funcție JavaScript al cărei nume începe cu 'use' și care poate apela alți hooks React. Custom hooks permit extragerea logicii cu state din componente și reutilizarea ei în mai multe locuri - fără a duplica cod.



VS Code: hooks/useSpeakeri.js - custom hook care encapsulează toată logica de fetch, loading, eroare și filtrare. Componenta Home.jsx importă și apelează hook-ul în 2 linii de cod în loc de 20.

```
// hooks/useSpeakeri.js - extras din Home.jsx pentru reutilizare
import { useState, useEffect } from 'react';
import { getSpeakeri } from '../api';

export function useSpeakeri(filtru = '') {
  const [speakeri, setSpeakeri] = useState([]);
  const [loading, setLoading]   = useState(true);
  const [eroare, setEroare]     = useState(null);

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

  const filtrati = speakeri.filter(sp =>
    sp.nume.toLowerCase().includes(filtru.toLowerCase())
  );

  return { speakeri: filtrati, loading, eroare };
}
```

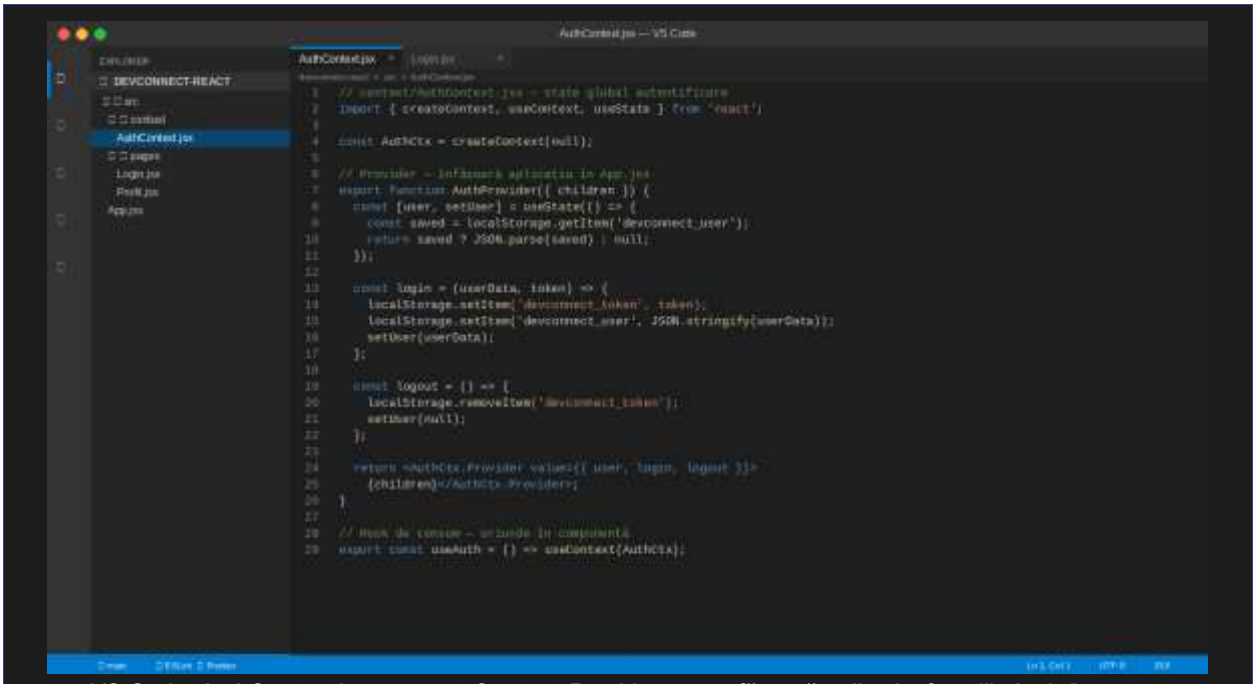
```
// Utilizare în orice componentă - logica este abstractizată:
function Home() {
  const [filtru, setFiltru] = useState('');
  const { speakeri, loading, eroare } = useSpeakeri(filtru);
  // ... JSX
}

// Același hook, altă componentă:
function AdminSpeakeri() {
  const { speakeri, loading } = useSpeakeri();
  // ... tabel admin
}
```

Regulile Hooks sunt obligatorii: (1) Apelează hooks DOAR la nivelul de top al funcției - nu în if, for, sau funcții imbricate. (2) Apelează hooks DOAR din funcții React (componente sau alte custom hooks). Aceste reguli permit React să mențină ordinea hook-urilor între render-uri.

8.2 Context API - State global fără prop drilling

Prop drilling înseamnă transmiterea unui prop prin mai multe niveluri de componente intermediare care nu au nevoie de el. Context API rezolvă asta: creezi un 'canal' de date care poate fi accesat direct de orice componentă din arbore, indiferent de adâncime.

Când să folosești Context vs useState local: useState local pentru state care aparține unei singure componente (un form, un dropdown deschis). Context pentru state care trebuie accesat în mai multe locuri: utilizatorul autentificat, tema (dark/light), limba, coșul de cumpărături.



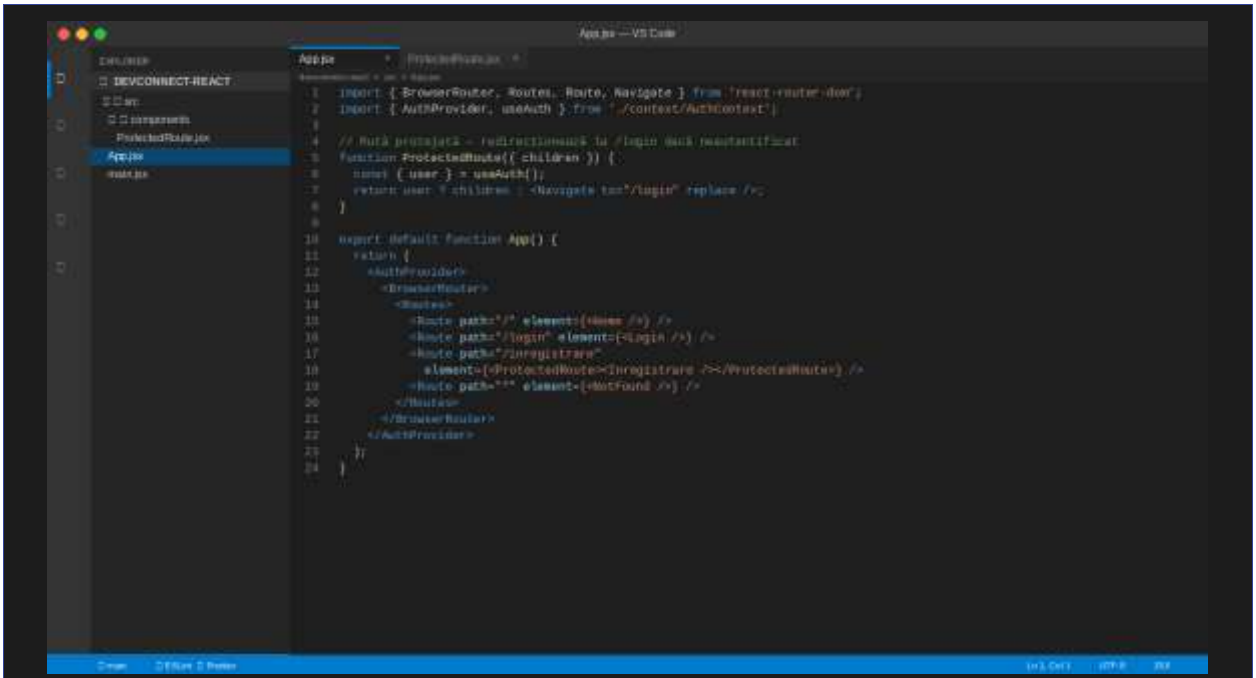
VS Code: AuthContext.jsx cu createContext, Provider care înfășoară aplicația, funcțiile login/logout cu localStorage, și hook-ul useAuth exportat pentru consum simplu în orice componentă. Observă inițializarea lazy din localStorage pentru persistență.

```
// 3 pași pentru Context API:

// PASUL 1: Creează contextul (context/AuthContext.jsx)
const AuthCtx = createContext(null);

// PASUL 2: Provider - înfășoară aplicația în App.jsx
```

```
export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const login  = (u, t) => { /* salvează user + token */ setUser(u); };
  const logout = ()     => { /* șterge din localStorage */ setUser(null); };
  return <AuthCtx.Provider value={{ user, login, logout }}>
    {children}
  </AuthCtx.Provider>;
}

// În App.jsx - înfășoară tot:
<AuthProvider>
  <BrowserRouter>...</BrowserRouter>
</AuthProvider>

// PASUL 3: Consumă în orice componentă
export const useAuth = () => useContext(AuthCtx);

// Utilizare - oriunde în aplicație, fără prop drilling:
function NavBar() {
  const { user, logout } = useAuth();
  return user ? <button onClick={logout}>Deconectare {user.nume}</button>
              : <Link to='/login'>Autentificare</Link>;
}
```

8.3 React Router v6 - Navigare în SPA

React Router permite navigarea între 'pagini' fără reîncărcarea paginii (Single Page Application). URL-ul se schimbă, componenta se înlocuiește, dar browser-ul nu face nicio cerere HTTP către server pentru HTML.



VS Code: App.jsx cu BrowserRouter, Routes și Route-uri configurate, inclusiv componenta ProtectedRoute care verifică autentificarea. Observă Navigate replace pentru redirect fără a adăuga în history browserului.

```
// Instalare:
npm install react-router-dom

// Hooks React Router utile în componente:
import { useNavigate, useParams, useLocation,
         Link, NavLink } from 'react-router-dom';
```

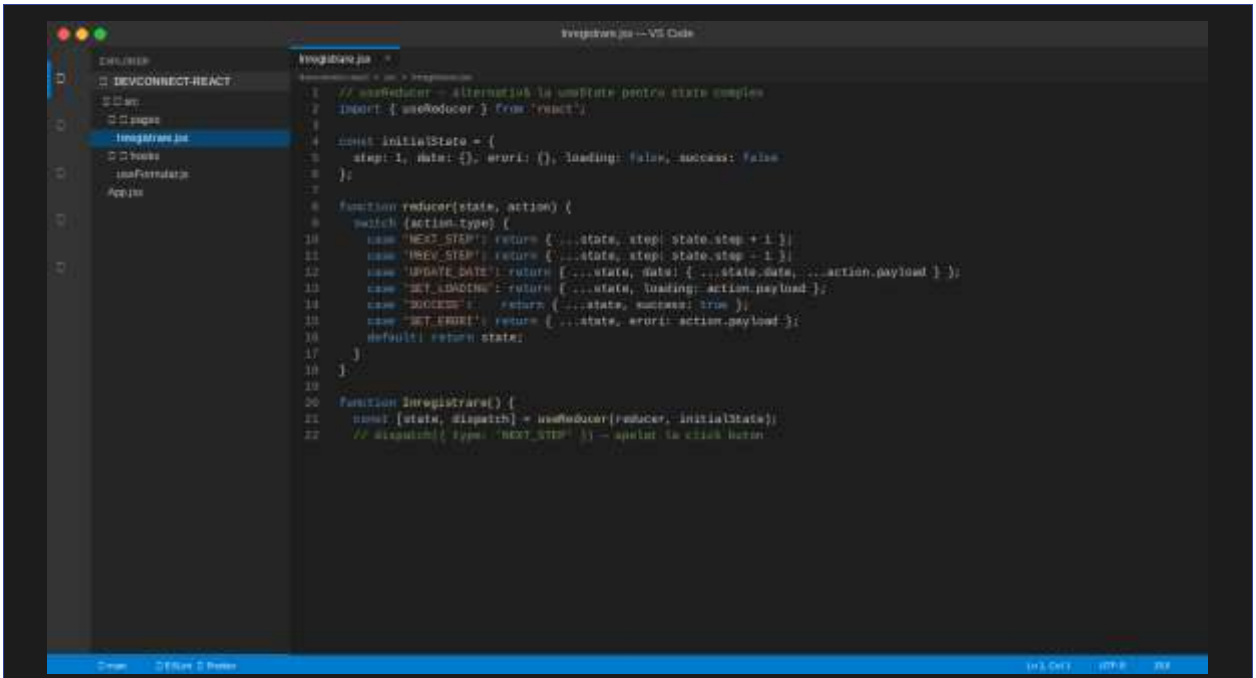
```
// useNavigate - redirect programatic
const navigate = useNavigate();
navigate('/dashboard');          // redirect simplu
navigate(-1);                    // înapoi (ca butonul Back)
navigate('/login', { replace: true }); // fără adăugare în history

// useParams - extrage parametrii din URL
// Ruta definită ca: <Route path='/speaker/:id' element={<SpeakerDetaliu/>}/>
const { id } = useParams();      // id = '42' din /speaker/42

// useLocation - informații despre URL curent
const location = useLocation();
// location.pathname = '/speaker/42'
// location.search   = '?tab=bio'
// location.state    = datele transmise prin navigate()

// Link vs NavLink:
<Link to='/speakeri'>Speakeri</Link>  // link simplu
<NavLink to='/speakeri'               // adaugă clasa 'active' automat
  className={({ isActive }) => isActive ? 'activ' : ''}
>Speakeri</NavLink>
```

Rute imbricate (Nested Routes) - layout comun

```
// Pattern: Layout cu outlet pentru rute imbricate
function Layout() {
  return (
    <div>
      <NavBar />            {/* apare pe toate paginile */}
      <main>
        <Outlet />          {/* componenta rutei active se randează aici */}
      </main>
      <Footer />
    </div>
  );
}

// În App.jsx:
<Routes>
  <Route element={<Layout />}>           {/* ruta-container cu layout */}
    <Route path='/'          element={<Home />} />
    <Route path='/speakeri'  element={<Speakeri />} />
    <Route path='/speaker/:id' element={<SpeakerDetaliu />} />
    <Route element={<ProtectedRoute />}> {/* rute protejate grupate */}
      <Route path='/inregistrare' element={<Inregistrare />} />
      <Route path='/profil'       element={<Profil />} />
    </Route>
  </Route>
  <Route path='*' element={<NotFound />} />
</Routes>
```

8.4 useReducer - State complex

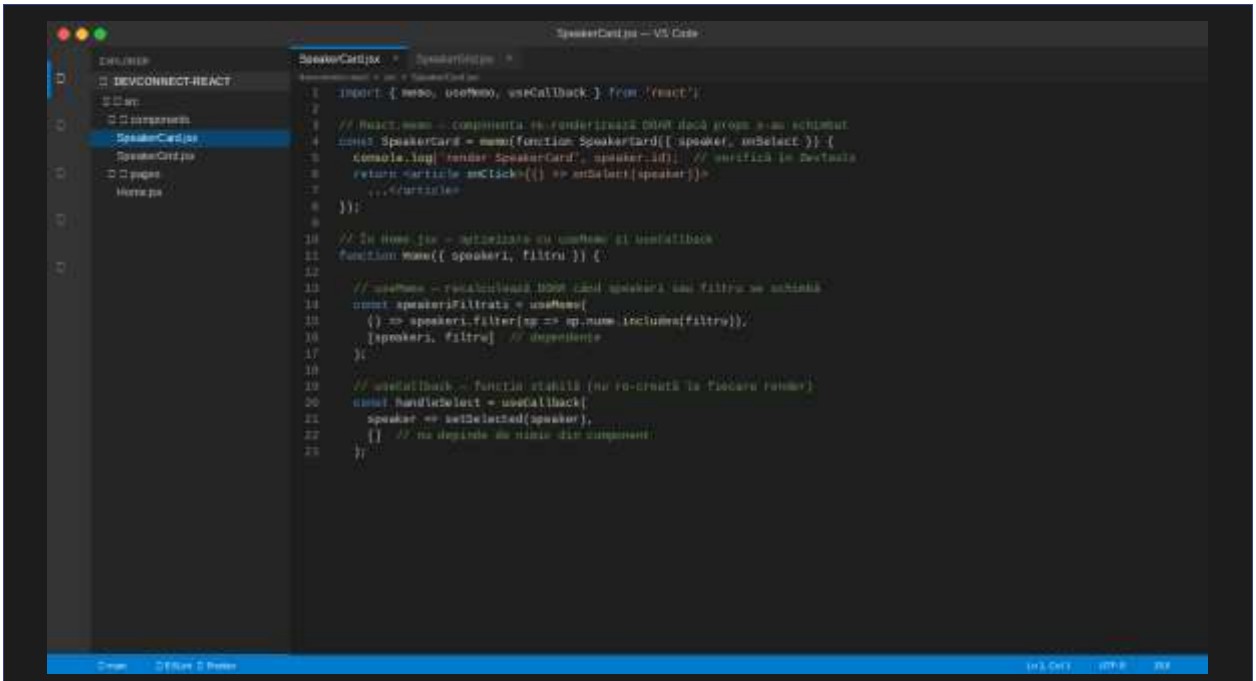
useReducer este alternativa la useState pentru state complex cu mai multe câmpuri interdependente sau tranziții de stare cu logică non-trivială. Urmează patternul Redux: stare → acțiune → stare nouă.



VS Code: Inregistrare.jsx cu useReducer pentru formularul multi-step. Reducer-ul gestionează toate tranzițiile posibile (NEXT_STEP, UPDATE_DATE, SUCCESS etc.) într-un loc central, cu logică clară și testabilă.

```
// Când useState e suficient vs când folosești useReducer:

// useState - ok pentru state simplu și independent:
const [open, setOpen]     = useState(false);
const [query, setQuery]   = useState('');

// useReducer - mai bun pentru:
// - state cu multe câmpuri (formular cu 10+ input-uri)
// - tranziții complexe (step wizard, mașină de stări)
// - logică de update dependentă de state anterior

// Exemplu simplu: coș de cumpărături
function cosReducer(state, action) {
  switch (action.type) {
    case 'ADAUGA':
      const existent = state.items.find(i => i.id === action.item.id);
      if (existent) return {
        ...state,
        items: state.items.map(i => i.id === action.item.id
          ? { ...i, qty: i.qty + 1 } : i)
      };
      return { ...state, items: [...state.items, {...action.item, qty:1}] };
    case 'STERGE':
      return { ...state, items: state.items.filter(i => i.id !== action.id) };
    case 'GOLESTE':
      return { items: [], total: 0 };
    default: return state;
  }
}

const [cos, dispatch] = useReducer(cosReducer, { items: [], total: 0 });
dispatch({ type: 'ADAUGA', item: { id: 1, nume: 'Bilet' } });
```

8.5 useMemo & useCallback - Optimizare performanță



VS Code: React.memo pe SpeakerCard previne re-renderizări inutile, useMemo pentru calculul filtrat costisitor, useCallback pentru stabilizarea funcției handler. Comentariile din cod arată când fiecare optimizare este necesară.

```
// Ordinea de gândire pentru optimizare React:
// 1. Profilează cu React DevTools Profiler - nu optimiza orb
// 2. Aplică React.memo pe componente care se renderizează des cu aceleași props
// 3. Folosește useMemo pentru calcule costisitoare (>5ms)
// 4. Folosește useCallback pentru funcții transmise ca props copiilor memoizați

// React.memo - memoizează componenta (nu re-renderizează dacă props identice)
const SpeakerCard = memo(function SpeakerCard({ speaker, onSelect }) {
  // renderizează DOAR când speaker sau onSelect se schimbă
  return <article onClick={() => onSelect(speaker)}>...</article>;
});

// useMemo - memoizează valoarea calculată
const speakeriFiltrati = useMemo(
  () => speakeri.filter(sp => sp.nume.includes(filtru))  // calcul costisitor
    .sort((a, b) => a.nume.localeCompare(b.nume)),
  [speakeri, filtru]  // recalculează DOAR când speakeri sau filtru se schimbă
);

// useCallback - memoizează funcția (referință stabilă)
const handleSelect = useCallback(
  (speaker) => setSelectedSpeaker(speaker),
  []  // funcție stabilă - nu se recreează la fiecare render
);
// Fără useCallback: SpeakerCard s-ar re-renderiza la fiecare render al
părintelui
// chiar dacă datele speaker nu s-au schimbat (referința funcției e nouă la
fiecare render)
```

> ⚠ Nu adăuga useMemo/useCallback pretutindeni în avans - are un cost (memorie pentru memoizare, comparație dependențe). Regula: aplică-le doar când React DevTools Profiler arată că o componentă se renderizează prea des sau un calcul durează vizibil.

8.6 Patterns React moderne (2024)

Compound Components

```
// Pattern: componente compuse care partajează state implicit
// Exemplu: Tab-uri în pagina DevConnect
function Tabs({ children, defaultTab = 0 }) {
  const [activ, setActiv] = useState(defaultTab);
  return (
    <TabContext.Provider value={{ activ, setActiv }}>
      <div className='tabs'>{children}</div>
    </TabContext.Provider>
  );
}
Tabs.Lista  = function TabLista({ children }) { return <ul>{children}</ul>; };
Tabs.Tab    = function Tab({ index, children }) {
  const { activ, setActiv } = useContext(TabContext);
  return <li className={activ===index?'activ':''}
onClick={()=>setActiv(index)}>{children}</li>;
};
Tabs.Panou  = function Panou({ index, children }) {
  const { activ } = useContext(TabContext);
  return activ === index ? <div>{children}</div> : null;
};

// Utilizare - API clar și flexibil:
<Tabs defaultTab={0}>
  <Tabs.Lista>
    <Tabs.Tab index={0}>Speakeri</Tabs.Tab>
    <Tabs.Tab index={1}>Program</Tabs.Tab>
  </Tabs.Lista>
  <Tabs.Panou index={0}><ListaSpeakeri /></Tabs.Panou>
  <Tabs.Panou index={1}><Program /></Tabs.Panou>
</Tabs>
```

8.8 DevConnect - Exemplificare Custom Hooks, Context și Router

Această secțiune construiește arhitectura avansată a DevConnect folosind toate conceptele din C8: custom hooks pentru logică reutilizabilă, AuthContext pentru autentificare globală, React Router pentru navigare.

Custom hook useSpeakeri - extragerea logicii din pagini

Fără custom hook, fiecare pagină care afișează speakeri (Speakeri.jsx, Home.jsx, AdminSpeakeri.jsx) ar duplica aceleași useState + useEffect. Extragem logica o dată:

```
// hooks/useSpeakeri.js - logică reutilizabilă pentru speakeri
import { useState, useEffect, useCallback } from 'react';
import { getSpeakeri } from '../api';

export function useSpeakeri(optiuni = {}) {
  const { filtruInitial = '', sesiuneInitiala = 'toate' } = optiuni;

  const [speakeri, setSpeakeri]   = useState([]);
  const [loading, setLoading]     = useState(true);
  const [eroare, setEroare]       = useState(null);
  const [cautare, setCautare]     = useState(filtruInitial);
  const [sesiune, setSesiune]     = useState(sesiuneInitiala);

  // Funcția de fetch - useCallback pentru referință stabilă
  const incarca = useCallback(() => {
    setLoading(true);
    setEroare(null);
    getSpeakeri()
```

```
        .then(setSpeakeri)
        .catch(err => setEroare(err.message))
        .finally(() => setLoading(false));
    }, []); // nu depinde de nimic → referință stabilă

    useEffect(() => { incarca(); }, [incarca]);

    // State derivat - calculat în hook, nu expus ca state
    const afisati = speakeri
      .filter(sp => sesiune === 'toate' || sp.sesiune === sesiune)
      .filter(sp => [sp.nume, sp.titlu, sp.companie]
        .join(' ').toLowerCase().includes(cautare.toLowerCase()));

    return {
      // Date
      speakeri: afisati,
      total: speakeri.length,
      // Stări
      loading, eroare,
      // Filtre + setteri
      cautare, setCautare,
      sesiune, setSesiune,
      // Acțiuni
      reincarca: incarca,
      resetFiltre: () => { setCautare(''); setSesiune('toate'); },
    };
  }

  // ── Utilizare în Speakeri.jsx ──────────────────────────
  function Speakeri() {
    const { speakeri, loading, eroare, cautare, setCautare,
            sesiune, setSesiune, reincarca, resetFiltre } = useSpeakeri();
    // ... JSX - componenta e curată, fără logică de fetch
  }

  // ── Utilizare în Home.jsx - doar primii 3 speakeri featured ─
  function SpeakeriFeatured() {
    const { speakeri, loading } = useSpeakeri();
    if (loading) return <LoadingSpinner />;
    return speakeri.slice(0, 3).map(sp => <SpeakerCard key={sp.id} {...sp} />);
  }
```

AuthContext complet - autentificare persistentă cu token JWT

AuthContext gestionează toată autentificarea aplicației: login, logout, persistența sesiunii, și expunerea utilizatorului curent tuturor componentelor:

```
  // context/AuthContext.jsx - implementare completă DevConnect
  import { createContext, useContext, useState, useCallback } from 'react';
  import { loginUser, refreshToken } from '../api';

  const AuthCtx = createContext(null);

  export function AuthProvider({ children }) {
    // useState cu inițializator lazy - citim localStorage O SINGURĂ DATĂ la
  montare
    // (nu la fiecare render ca useState(JSON.parse(...)))
    const [user, setUser] = useState(() => {
      try {
        const saved = localStorage.getItem('dc_user');
        return saved ? JSON.parse(saved) : null;
      } catch { return null; } // JSON corupt → deconectat
    });

    const login = useCallback(async (email, parola) => {
      // loginUser face POST /auth/login → { user, accessToken }
```

```
    const { user: userData, accessToken } = await loginUser({ email, parola });

    // Persistăm token în memory (nu localStorage - securitate mai bună)
    // Refresh token e în cookie HttpOnly setat de server (Lab 12)
    sessionStorage.setItem('dc_token', accessToken);
    localStorage.setItem('dc_user', JSON.stringify(userData));
    setUser(userData);
    return userData;
  }, []);

  const logout = useCallback(() => {
    sessionStorage.removeItem('dc_token');
    localStorage.removeItem('dc_user');
    setUser(null);
    // În Lab 12: POST /auth/logout să invalideze refresh token pe server
  }, []);

  const value = {
    user,
    login,
    logout,
    esteAutentificat: Boolean(user),
    // Rolul utilizatorului - util pentru UI condițional
    rol: user?.rol ?? 'anonim', // 'participant' | 'speaker' | 'admin'
    getToken: () => sessionStorage.getItem('dc_token'),
  };

  return <AuthCtx.Provider value={value}>{children}</AuthCtx.Provider>;
}

// Hook pentru consum - cu validare că e în provider
export function useAuth() {
  const ctx = useContext(AuthCtx);
  if (!ctx) throw new Error('useAuth() trebuie folosit în interiorul
<AuthProvider>');
  return ctx;
}
```

React Router - structura completă a rutelor DevConnect

```
// App.jsx - configurarea completă a rutelor DevConnect
import { BrowserRouter, Routes, Route, Navigate, Outlet } from 'react-router-
dom';
import { useAuth } from './context/AuthContext';

// ─── Componenta ProtectedRoute ──────────────────────────────
// Redirectează la /login dacă utilizatorul nu e autentificat
// Transmite locația curentă pentru redirect înapoi după login
function ProtectedRoute({ roles = [] }) {
  const { esteAutentificat, rol } = useAuth();
  const location = useLocation();

  if (!esteAutentificat) {
    // Salvăm unde voia să ajungă - după login îl redirecționăm înapoi
    return <Navigate to='/login' state={{ from: location }} replace />;
  }

  if (roles.length > 0 && !roles.includes(rol)) {
    return <Navigate to='/acces-refuzat' replace />;
  }

  return <Outlet />; // randează ruta copil dacă autentificat + rol ok
}

// ─── App.jsx - rutele complete ──────────────────────────────
```

```
export default function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <Routes>

          {/* Layout cu NavBar + Footer pentru toate rutele publice */}
          <Route element={<Layout />}>
            <Route path='/'             element={<Home />} />
            <Route path='/speakeri'     element={<Speakeri />} />
            <Route path='/speakeri/:id' element={<SpeakerDetaliu />} />
            <Route path='/workshops'    element={<Workshops />} />
            <Route path='/program'      element={<Program />} />
            <Route path='/login'        element={<Login />} />

            {/* Rute protejate - necesită autentificare */}
            <Route element={<ProtectedRoute />}>
              <Route path='/inregistrare' element={<Inregistrare />} />
              <Route path='/profil'       element={<Profil />} />
            </Route>

            {/* Rute admin - necesită rol 'admin' */}
            <Route element={<ProtectedRoute roles={['admin']} />}>
              <Route path='/admin'           element={<AdminDashboard />} />
              <Route path='/admin/speakeri'  element={<AdminSpeakeri />} />
              <Route path='/admin/inscrisi'  element={<AdminInscrisi />} />
            </Route>
          </Route>

          <Route path='*' element={<NotFound />} />
        </Routes>
      </BrowserRouter>
    </AuthProvider>
  );
}
```

NavBar cu state reactiv - autentificare + navigare activă

```
// components/NavBar.jsx - navbar complet DevConnect
import { Link, NavLink, useNavigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

const LINKURI_NAV = [
  { to: '/',           label: 'Acasă' },
  { to: '/speakeri',   label: 'Speakeri' },
  { to: '/workshops',  label: 'Workshops' },
  { to: '/program',    label: 'Program' },
];

function NavBar() {
  const { user, esteAutentificat, logout, rol } = useAuth();
  const navigate = useNavigate();
  const [menuOpen, setMenuOpen] = useState(false); // mobile hamburger

  const handleLogout = () => {
    logout();
    navigate('/', { replace: true });
  };

  return (
    <header className='navbar'>
      <Link to='/' className='navbar-logo'>DevConnect 2026</Link>

      {/* Mobile: buton hamburger */}
      <button className='hamburger' onClick={() => setMenuOpen(m => !m)}
              aria-label='Deschide meniu' aria-expanded={menuOpen}>
```

```
          ☰
        </button>

        <nav className={`navbar-nav ${menuOpen ? 'deschis' : ''}`}>
          {LINKURI_NAV.map(({ to, label }) => (
            <NavLink key={to} to={to} end
              className={({ isActive }) => isActive ? 'nav-link activ' : 'nav-
link'}
              onClick={() => setMenuOpen(false)}
            >
              {label}
            </NavLink>
          ))}

          {/* Secțiunea de autentificare - condiționat */}
          {esteAutentificat ? (
            <div className='nav-user'>
              {rol === 'admin' && (
                <Link to='/admin' className='nav-link admin-badge'>Admin ⚙</Link>
              )}
              <span className='user-greeting'>Salut, {user.prenume}!</span>
              <Link to='/profil' className='nav-link'>Profilul meu</Link>
              <button onClick={handleLogout} className='btn-logout'>
                Deconectare
              </button>
            </div>
          ) : (
            <div className='nav-auth'>
              <Link to='/login' className='btn-login'>Autentificare</Link>
              <Link to='/inregistrare' className='btn-primary'>Înregistrare</Link>
            </div>
          )}
        </nav>
      </header>
    );
  }

  export default NavBar;
```

useReducer pentru formularul de înregistrare multi-step

Formularul de înregistrare DevConnect are 3 pași: date personale → selectare workshop → confirmare. Folosim useReducer pentru a gestiona toată starea într-un singur loc:

```
  // Mașina de stări pentru formularul multi-step
  // Starea completă: { step, date, erori, loading, success }

  const INITIAL_STATE = {
    step: 0,
    date: {
      // Pas 1 - Date personale
      nume: '', prenume: '', email: '', telefon: '',
      // Pas 2 - Workshop
      workshopId: '', tipBilet: 'standard',
      // Pas 3 - Preferințe
      optiuneDietar: '', newsletter: false,
    },
    erori: {}, // { camp: 'mesaj eroare' }
    loading: false,
    success: null, // null | { id, mesaj }
    eroareGenerala: null,
  };

  function reducer(state, { type, payload }) {
    switch (type) {
      case 'NEXT_STEP':
```

```
        return { ...state, step: Math.min(state.step + 1, 2), erori: {} };
      case 'PREV_STEP':
        return { ...state, step: Math.max(state.step - 1, 0) };
      case 'UPDATE_DATE':
        // payload: { camp: valoare } - actualizare parțială a date
        return { ...state, date: { ...state.date, ...payload }, erori: {} };
      case 'SET_ERORI':
        return { ...state, erori: payload };
      case 'SUBMIT_START':
        return { ...state, loading: true, eroareGenerala: null };
      case 'SUBMIT_SUCCESS':
        return { ...state, loading: false, success: payload };
      case 'SUBMIT_FAIL':
        return { ...state, loading: false, eroareGenerala: payload };
      default: return state;
    }
  }

  // ─── Componenta principală ──────────────────────────────
  function Inregistrare() {
    const [state, dispatch] = useReducer(reducer, INITIAL_STATE);
    const navigate = useNavigate();

    const handleSubmitFinal = async () => {
      dispatch({ type: 'SUBMIT_START' });
      try {
        const rezultat = await inregistreazaParticipant(state.date);
        dispatch({ type: 'SUBMIT_SUCCESS', payload: rezultat.inregistrare });
        // Redirect după 3 secunde
        setTimeout(() => navigate('/'), 3000);
      } catch (err) {
        dispatch({ type: 'SUBMIT_FAIL', payload: err.message });
      }
    };

    if (state.success) return (
      <div className='success-box'>
        <h2>☑ Înregistrare confirmată!</h2>
        <p>ID: #{state.success.id} - Email de confirmare a fost trimis.</p>
        <p>Vei fi redirecționat la pagina principală în 3 secunde...</p>
      </div>
    );

    return (
      <div className='formular-inregistrare'>
        {/* Indicator progres pași */}
        <StepIndicator pasi={['Date personale', 'Workshop', 'Confirmare']}
                       activ={state.step} />

        {/* Eroare generală (de la API) */}
        {state.eroareGenerala && (
          <div className='eroare-generala'>✕ {state.eroareGenerala}</div>
        )}

        {/* Pașii formularului - randăm doar pasul activ */}
        {state.step === 0 && (
          <PasDatePersonale state={state} dispatch={dispatch} />
        )}
        {state.step === 1 && (
          <PasWorkshop state={state} dispatch={dispatch} />
        )}
        {state.step === 2 && (
          <PasConfirmare state={state} dispatch={dispatch}
                         onSubmit={handleSubmitFinal} />
        )}
      </div>
    );
  }
```

De ce useReducer în loc de useState multiplu? Dacă am folosi useState separat pentru fiecare câmp (step, loading, erori, date, success), am risca actualizări inconsistente: de exemplu, a seta loading=true și a uita erori=null simultan. useReducer garantează că fiecare acțiune produce o tranziție atomică și consistentă a întregii stări.

8.9 Referințe Curs 8

- react.dev - Hooks Reference (documentație oficială)
- react.dev - Managing State (ghid complet state)
- React Router v6 - documentație oficială
- TanStack Query (alternativă modernă la useEffect+fetch)
- Patterns.dev - Design patterns React moderne
- Bulletproof React - arhitectură proiecte React mari

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.