

React — dane z API

Czas: 2 × 45 min

praca indywidualna

Cel: Pobieranie danych z API w aplikacji React (useEffect + fetch) z obsługą stanu ładowania i błędu, filtrowanie i sortowanie po stronie klienta oraz przygotowanie wersji produkcyjnej.

Po zajęciach potrafisz:

- pobiera dane z API w hooku useEffect z poprawną tablicą zależności i sprzątaniem (AbortController)
- obsługuje w interfejsie trzy stany: ładowanie, błąd i dane
- filtruje i sortuje listę na podstawie stanu formularza kontrolowanego bez ponownego pobierania danych
- dzieli interfejs na komponenty z props
- buduje wersję produkcyjną (`npm run build`) i uruchamia folder dist

Sprzęt i materiały

- komputer z Windows 11, XAMPP (Apache) i Node.js LTS
- folder `r2_23_api` z Pliki_do_zadan (plik `api.php` i dane `gry.json`)
- projekt React z zadania 22 lub nowy projekt Vite

Oprogramowanie

- **Node.js LTS** — zawiera npm; sprawdź `node -v` i `npm -v`
nodejs.org
- **Visual Studio Code** — terminal wbudowany: Terminal → New Terminal
code.visualstudio.com
- **XAMPP** — Apache serwuje API pod `http://localhost/r2_23_api/api.php`
apachefriends.org/download.html
- **React DevTools (opcjonalnie)** — podgląd stanu komponentów w F12
react.dev/learn/react-developer-tools

Przygotowanie przed zajęciami

- Przypomnij sobie z prezentacji 2.15 hooki useState i useEffect oraz z zadania 4 obsługę fetch i async/await.
- Upewnij się, że projekt z zadania 22 uruchamia się poleceniem `npm run dev`.

Bezpieczeństwo

- Nie umieszczaj w kodzie frontendu haseł ani kluczy API — wszystko, co trafia do folderu dist, może przeczytać każdy użytkownik.
- Wyświetlaj dane w JSX zwykłymi wyrażeniami `{...}` (React sam je escapuje); nie używaj dangerouslySetInnerHTML.

Przebieg zadania

1 Przygotowanie API i projektu

API zwraca tablicę obiektów: id, nazwa, kategoria, min_graczy, max_graczy, czas_min, wiek_od, ocena, cena, dostepna. Parametry testowe: `?opoznienie=2` (odpowiedź po 2 s) i `?blad=1` (kod 500).

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

1. Skopiuj folder `r2_23_api` do `C:\xampp\htdocs`, uruchom Apache i otwórz `http://localhost/r2_23_api/api.php`. Sprawdź w F12 → Network kod odpowiedzi i nagłówek Content-Type.
2. Utwórz projekt poleceniami z ramki, usuń przykładową zawartość `App.jsx` i `App.css`, ustaw `lang="pl"` i tytuł strony w `index.html`.
3. Utwórz plik `.env` z wierszem `VITE_API_URL=http://localhost/r2_23_api/api.php` i odczytaj adres w kodzie przez `import.meta.env.VITE_API_URL`. Po zmianie `.env` uruchom ponownie `npm run dev`.

2 Pobieranie danych w useEffect

Stan aplikacji: `gry` (tablica), `ladowanie` (true/false), `blad` (null albo tekst).

1. W `App.jsx` utwórz trzy stany `useState` i w `useEffect` z pustą tablicą zależności pobierz dane funkcją `async` (`fetch` + `await odp.json()`). Wyświetl na razie same nazwy gier w liście `<ul>` z atrybutem `key={g.id}`.
2. Sprawdź `odp.ok` i przy błędzie HTTP zgłoś wyjątek z kodem odpowiedzi; w bloku `catch` zapisz komunikat do stanu `blad`, a w `finally` wyłącz `ladowanie`.
3. Wyświetlaj warunkowo: „Ładowanie danych...” (`role="status"`), komunikat błędu (`role="alert"`) albo listę. Sprawdź oba stany, dopisując do adresu API `?opoznienie=2`, a potem `?blad=1`; zrób zrzuty.
4. Dodaj przycisk „Spróbuj ponownie” przy błędzie: zwiększa licznik w stanie `proba`, który dopisz do tablicy zależności `useEffect`.
5. Dodaj sprzątanie efektu: utwórz `AbortController`, przekaż `signal` do `fetch` i zwróć z efektu funkcję `() => kontroler.abort()`. Wyjaśnij w sprawozdaniu, dlaczego w trybie `StrictMode` w F12 → Network widać dwa żądania.

3 Komponenty i props

1. Utwórz komponent `GraKarta` (folder `src/components`) wyświetlający kartę gry: kategorię, nazwę, liczbę graczy, czas, wiek, cenę w zł (`Intl.NumberFormat('pl-PL', {style: 'currency', currency: 'PLN'})`) i ocenę.
2. Gry niedostępne (`dostepna: false`) wyróżnij inną klasą CSS i dopiskiem „chwilowo niedostępna”. Wyświetl karty w siatce CSS Grid (auto-fill, min. 230 px).

4 Filtrowanie i sortowanie po stronie klienta

Dane pobierasz raz — filtry tylko przeliczają widoczną listę przy każdym renderowaniu. Nie twórz osobnego stanu na przefiltrowaną tablicę.

1. Utwórz komponent `Filtry` z formularzem kontrolowanym: pole wyszukiwania po nazwie (bez względu na wielkość liter), lista kategorii (zbudowana z danych przez `new Set`, plus opcja „wszystkie”), lista sortowania i pole wyboru „tylko dostępne”. Stany trzymaj w App i przekazuj przez props.
2. Oblicz tablicę `widoczne` łańcuchem `filter(...)` i sortowaniem kopii tablicy (`toSorted` albo `[...tab].sort`): nazwa A–Z (`localeCompare(b, 'pl')`), cena rosnąco, cena malejąco, najlepiej oceniane.
3. Wyświetl licznik „Wyświetlono X z Y gier” i komunikat, gdy żadna gra nie spełnia warunków. Zapisz w sprawozdaniu wynik dla: szukaj „ko”; kategoria rodzinna + tylko dostępne + cena rosnąco.

5 Wersja produkcyjna

1. W `vite.config.js` dodaj `base: './'`, uruchom `npm run build` i zapisz rozmiar plików JS i CSS z komunikatu Vite.
2. Uruchom `npm run preview` i sprawdź działanie wersji z folderu `dist`. Następnie skopiuj zawartość `dist` do `htdocs/katalog` i otwórz `http://localhost/katalog/`.
3. Otwórz plik `dist/index.html` bezpośrednio z dysku (dwuklik) i zanotuj, co widać w konsoli F12. Wyjaśnij, dlaczego wersję produkcyjną trzeba serwować przez serwer HTTP.
4. Dodaj do `.gitignore` foldery `node_modules` i `dist` (jeśli ich brak) i spakuj projekt bez `node_modules` do folderu `nazwisko_imie`.

W sprawozdaniu umieść

- ☐ zrzuty: stan ładowania, stan błędu, lista z filtrami w działaniu
- ☐ fragment kodu `useEffect` z obsługą błędu i `AbortController`
- ☐ wyniki filtrów z kroku 4 (liczba gier i kolejność)
- ☐ rozmiary plików z `npm run build` i obserwacja z otwarcia `dist` z dysku
- ☐ projekt bez `node_modules` i odpowiedzi na pytania

Pytania kontrolne

1. Co się stanie, jeśli w `useEffect` pominiesz tablicę zależności, a w efekcie ustawiasz stan?
2. Dlaczego `fetch` nie zgłasza wyjątku przy kodzie 500 i jak to obsłużyłeś?
3. Dlaczego listy przefiltrowanej nie trzeba przechowywać w osobnym stanie?
4. Po co nagłówek `Access-Control-Allow-Origin` w `api.php`, skoro Vite i Apache działają na tym samym komputerze?

Kryteria oceny

Kryterium	Punkty
Pobieranie danych w <code>useEffect</code> , sprawdzenie odp.ok, <code>AbortController</code>	4
Trzy stany interfejsu (ładowanie, błąd z przyciskiem ponowienia, dane) — zrzuty	3
Komponenty <code>GraKarta</code> i <code>Filtry</code> z props, <code>key</code> w liście, wyróżnienie gier niedostępnych	3
Filtrowanie (tekst, kategoria, dostępność) i 4 sposoby sortowania, licznik wyników	4
<code>Build</code> , <code>preview</code> i działanie z <code>htdocs</code> , <code>base './'</code> , <code>.gitignore</code>	3
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik `.docx` o tym samym numerze).