

# Zestaw Zadań: Podstawy React JS

## Informacje Ogólne

<b>Materiał źródłowy:</b>	Wprowadzenie do React JS - komponenty, JSX, props, state, hooks (useState, useEffect), obsługa zdarzeń, formularze
<b>Poziom:</b>	Mieszany (podstawowy → zaawansowany)
<b>Szacowany czas:</b>	6-7 godzin (wszystkie zadania)
<b>Typ zestawu:</b>	Powiązane - zadania budują wspólną aplikację "Menedżer Zadań"
<b>Wymagania wstępne:</b>	Znajomość HTML, CSS, JavaScript (ES6+)

## Wprowadzenie do React

### Czym jest React?

React to biblioteka JavaScript do budowania interfejsów użytkownika. Pozwala tworzyć aplikacje z małych, izolowanych fragmentów kodu zwanych komponentami. React używa wirtualnego DOM do efektywnego aktualizowania tylko tych części strony, które rzeczywiście się zmieniły.

Główne koncepcje React to: komponenty (funkcyjne i klasowe), JSX (składnia łącząca JavaScript z HTML), props (przekazywanie danych w dół), state (stan wewnętrzny komponentu) oraz hooks (funkcje pozwalające używać stanu i innych funkcji React w komponentach funkcyjnych).

### Najczęstsze problemy i pułapki

#### 1. JSX to nie HTML

JSX wygląda jak HTML, ale jest komplikowany do JavaScript. Używaj className zamiast class i htmlFor zamiast for.

 **Ostrzeżenie:** W JSX używaj className zamiast class i htmlFor zamiast for - to słowa zastrzeżone w JavaScript.

#### 2. Mutowanie stanu

Nigdy nie modyfikuj stanu bezpośrednio - zawsze używaj funkcji setter z useState.

 **Zasada:** Stan w React musi być niemutowalny (immutable). Zawsze twórz nowe obiekty/tablice zamiast modyfikować istniejące.

#### 3. Brakujące klucze w listach

Każdy element renderowany w pętli musi mieć unikalny atrybut key.

 **Wskazówka:** Używaj unikalnych ID jako kluczy, nie indeksów tablicy - indeksy mogą powodować błędy przy sortowaniu/filtrowaniu.

#### 4. useEffect i zależności

Pusta tablica zależności [] oznacza wykonanie tylko raz przy montowaniu. Brak tablicy oznacza wykonanie przy każdym renderze.

 **Zasada:** Zawsze dodawaj tablicę zależności do useEffect i uwzględniaj wszystkie zmienne używane wewnętrz.

## Zadanie 1: Pierwszy komponent React

<b>Poziom trudności:</b>	1  (Podstawowy)
<b>Szacowany czas:</b>	40 minut
<b>Punktacja:</b>	15 punktów
<b>Typ:</b>	Praktyczne
<b>Powiązanie:</b>	Niezależne - początek projektu "Menedżer Zadań"

### Cel dydaktyczny

Zrozumienie struktury projektu React, składni JSX oraz tworzenia pierwszych komponentów funkcyjnych.

### Wstęp teoretyczny

#### JSX - JavaScript XML

JSX to rozszerzenie składni JavaScript, które wygląda jak HTML, ale jest komplikowane do zwykłego JavaScript. Główne różnice: className zamiast class, htmlFor zamiast for, atrybuty w camelCase (onClick, onChange), wyrażenia JS w nawiasach klamrowych {zmienna}.

#### Komponenty funkcyjne

Komponent to funkcja zwracająca JSX. Nazwa komponentu musi zaczynać się wielką literą (PascalCase).

#### Polecenie

Stwórz nowy projekt React i zbuduj podstawową strukturę aplikacji "Menedżer Zadań":

#### Część A - Konfiguracja projektu (4 pkt)

- Utwórz nowy projekt React za pomocą Vite: `npm create vite@latest task-manager --template react`
- Usuń niepotrzebną zawartość z App.css i usuń folder assets
- Wyczyść plik App.jsx - zostaw tylko pusty komponent zwracający `<div>Hello React!</div>`
- Sprawdź czy aplikacja działa w przeglądarce (<http://localhost:5173>)

#### Część B - Struktura folderów (2 pkt)

- Stwórz folder src/components/ na komponenty aplikacji

#### Część C - Komponenty aplikacji (9 pkt)

Stwórz następujące komponenty (na razie bez logiki, tylko statyczny JSX):

- components/Header.jsx (3 pkt) - Ikona/emoji  i tytuł "Menedżer Zadań", aktualna data, podstawowe style
- components/TaskItem.jsx (3 pkt) - Element `<li>` z checkbox, tekst zadania, przycisk "Usuń"
- components/TaskList.jsx (2 pkt) - Kontener `<ul>` z 2-3 komponentami TaskItem
- App.jsx (1 pkt) - Złożenie całości w `<div className="app">`

#### Materiały/Zasoby potrzebne

- Node.js (v18+) i npm
- Edytor kodu (VS Code z rozszerzeniem ES7+ React)

- Dokumentacja React: react.dev/learn

### Kryteria oceny

- Poprawna konfiguracja projektu Vite, aplikacja się uruchamia (4 pkt)
- Prawidłowa struktura folderów (2 pkt)
- Komponent Header z prawidłowym JSX (3 pkt)
- Komponent TaskItem z checkboxem i przyciskiem (3 pkt)
- Komponent TaskList używający TaskItem (2 pkt)
- Prawidłowe importy/eksporty i złożenie w App.jsx (1 pkt)

### Wskazówki

- Każdy komponent w osobnym pliku z rozszerzeniem .jsx
- Używaj export default NazwaKomponentu na końcu pliku
- Import: import Header from './components/Header'
- Jeśli widzisz błędy w konsoli przeglądarki (F12) - przeczytaj je, często podpowiadają rozwiązanie

## Zadanie 2: Props - przekazywanie danych

<b>Poziom trudności:</b>	2 🟡 (Średniozaawansowany)
<b>Szacowany czas:</b>	45 minut
<b>Punktacja:</b>	20 punktów
<b>Typ:</b>	Praktyczne
<b>Powiązanie:</b>	Rozszerza Zadanie 1 (struktura komponentów)

### Cel dydaktyczny

Opanowanie przekazywania danych między komponentami za pomocą props, destruktywizacji oraz wykorzystania props.children.

### Polecenie

Rozbuduj aplikację z Zadania 1, dodając dynamiczne przekazywanie danych przez props:

#### Część A - Props w TaskItem (6 pkt)

1. Zmodyfikuj komponent TaskItem, aby przyjmował props: id, title, completed, priority
2. Wyświetl dane z props: tytuł zadania, wizualne oznaczenie priorytetu, checkbox zaznaczony jeśli completed === true, przekreślony tekst dla ukończonych

#### Część B - Props w TaskList (6 pkt)

1. Stwórz tablicę przykładowych zadań w App.jsx (min. 3 zadania z różnymi priorytetami)
2. Przekaż tablicę do TaskList jako prop tasks
3. W TaskList użyj metody .map() do wyrenderowania komponentów TaskItem
4. Dodaj obsługę pustej listy - wyświetl komunikat gdy tasks.length === 0

#### Część C - Props children i kompozycja (5 pkt)

1. Stwórz komponent Card.jsx - uniwersalny wrapper przyjmujący children, opcjonalny title i className

- Użyj komponentu Card do opakowania TaskList

#### Część D - Domyślne props (3 pkt)

Dodaj domyślne wartości dla props w TaskItem: priority domyślnie "medium", completed domyślnie false

#### Kryteria oceny

- TaskItem przyjmuje i wyświetla wszystkie props (4 pkt)
- Wizualne oznaczenie priorytetu i statusu (2 pkt)
- Prawidłowe użycie .map() z kluczami (4 pkt)
- Obsługa pustej listy (2 pkt)
- Komponent Card z children (3 pkt)
- Domyślne wartości props (2 pkt)
- Czytelność kodu i organizacja (3 pkt)

#### Wskazówki

- Destrukturyzuj props w parametrach funkcji: function TaskItem({ title, completed })
- Klucz w .map() powinien być unikalny - użyj id, nie index
- Domyślne wartości: function Component({ priority = "medium" })
- props.children to specjalny prop - zawartość między tagami komponentu

### Zadanie 3: State i obsługa zdarzeń

<b>Poziom trudności:</b>	2  (Średniozaawansowany)
<b>Szacowany czas:</b>	60 minut
<b>Punktacja:</b>	25 punktów
<b>Typ:</b>	Praktyczne
<b>Powiązanie:</b>	Rozszerza Zadanie 2 (props i struktura danych)

#### Cel dydaktyczny

Opanowanie hooka useState do zarządzania stanem aplikacji oraz obsługi zdarzeń użytkownika (kliknięcia, zmiany).

#### Polecenie

Dodaj interaktywność do aplikacji Menedżer Zadań:

#### Część A - Stan listy zadań (8 pkt)

- Przenieś tablicę zadań do stanu w App.jsx używając useState
- Zaimplementuj funkcję toggleTask(id) - zmienia completed na przeciwny
- Zaimplementuj funkcję deleteTask(id) - usuwa zadanie z listy
- Podłącz funkcje: checkbox wywołuje toggleTask, przycisk "Usuń" wywołuje deleteTask

#### Część B - Filtrowanie zadań (7 pkt)

- Dodaj stan filter z wartościami: "all", "active", "completed"
- Stwórz komponent FilterButtons.jsx z trzema przyciskami i wizualnym wyróżnieniem aktywnego filtra

- Zaimplementuj filtrowanie listy przed renderowaniem

### Część C - Statystyki (5 pkt)

- Stwórz komponent TaskStats.jsx wyświetlający: łączną liczbę zadań, liczbę ukończonych, pozostałych, procent ukończenia
- Obliczenia wykonuj na podstawie props tasks (nie twórz osobnego stanu)

### Część D - Zmiana priorytetu (5 pkt)

- Dodaj do TaskItem select/dropdown z opcjami priorytetu
- Zaimplementuj funkcję changePriority(id, newPriority) w App.jsx

### Kryteria oceny

- Prawidłowe użycie useState dla listy zadań (3 pkt)
- Funkcja toggleTask działa poprawnie (3 pkt)
- Funkcja deleteTask działa poprawnie (2 pkt)
- Filtrowanie zadań (5 pkt)
- Wizualne wyróżnienie aktywnego filtra (2 pkt)
- Komponent statystyk (4 pkt)
- Zmiana priorytetu (4 pkt)
- Niemutowalność stanu (2 pkt)

### Wskazówki

 **Ważne:** Nigdy nie mutuj stanu bezpośrednio! Użyj: setTasks(tasks.map(task => task.id === id ? { ...task, completed: !task.completed } : task))

- Przekazując funkcję z argumentem: onClick={() => deleteTask(id)}
- Stan powinien być "single source of truth" - statystyki obliczaj z tasks

## Zadanie 4: Formularze w React

Poziom trudności:	3  (Zaawansowany)
Szacowany czas:	60 minut
Punktacja:	25 punktów
Typ:	Praktyczne
Powiezanie:	Rozszerza Zadanie 3 (stan i zdarzenia)

### Cel dydaktyczny

Opanowanie kontrolowanych komponentów formularzy, walidacji danych wejściowych oraz obsługi złożonych formularzy z wieloma polami.

### Polecenie

Dodaj formularz do tworzenia nowych zadań oraz funkcję edycji istniejących:

### Część A - Formularz dodawania zadania (10 pkt)

- Stwórz komponent TaskForm.jsx z polami: input tekstowy na tytuł, select na priorytet, przycisk "Dodaj zadanie"
- Użyj kontrolowanych komponentów (stan dla każdego pola)

11. Obsłuż wysłanie formularza: onSubmit z e.preventDefault(), wywołaj funkcję addTask, wyczyść formularz po dodaniu
12. Generuj unikalne id dla nowych zadań (użyj Date.now() lub crypto.randomUUID())

### Część B - Walidacja formularza (7 pkt)

5. Dodaj walidację: tytuł wymagany (min. 3 znaki), max 100 znaków, licznik znaków
6. Wyświetlaj komunikaty błędów pod polem, czerwona ramka dla nieprawidłowego pola
7. Przycisk "Dodaj" nieaktywny gdy formularz nieprawidłowy

### Część C - Edycja zadania (8 pkt)

5. Dodaj tryb edycji do TaskItem: przycisk "Edytuj" przełącza w tryb edycji
6. W trybie edycji: input zamiast tekstu, przyciski "Zapisz" i "Anuluj"
7. Zaimplementuj stan lokalny w TaskItem dla trybu edycji
8. Funkcja updateTask(id, newTitle) w App.jsx
9. Obsłuż skróty: Enter zapisuje, Escape anuluje

### Kryteria oceny

- Kontrolowany formularz dodawania (4 pkt)
- Obsługa onSubmit i czyszczenie formularza (3 pkt)
- Generowanie unikalnego id (1 pkt)
- Walidacja tytułu (3 pkt)
- Wyświetlanie błędów walidacji (2 pkt)
- Blokowanie nieprawidłowego formularza (2 pkt)
- Tryb edycji w TaskItem (4 pkt)
- Obsługa klawiatury (Enter/Escape) (2 pkt)
- Funkcja updateTask (2 pkt)
- UX i obsługa błędów (2 pkt)

### Wskazówki

- Kontrolowany komponent: wartość inputa ZAWSZE z state, zmiana przez onChange
- Walidacja: stwórz funkcję validate() zwracającą obiekt błędów
- Skrót klawiaturowy: onKeyDown={({e}) => e.key === "Enter" && handleSave()}}

## Zadanie 5: useEffect i pobieranie danych

<b>Poziom trudności:</b>	3  (Zaawansowany)
<b>Szacowany czas:</b>	70 minut
<b>Punktacja:</b>	30 punktów
<b>Typ:</b>	Praktyczne
<b>Powiązanie:</b>	Rozszerza Zadanie 4 (formularze i walidacja)

### Cel dydaktyczny

Opanowanie hooka useEffect do obsługi efektów ubocznych: pobieranie danych z API, zapisywanie do localStorage, oraz zarządzanie cyklem życia komponentu.

## Polecenie

Dodaj persystencję danych i integrację z zewnętrznym API:

### Część A - localStorage (8 pkt)

13. Zapisuj listę zadań do localStorage przy każdej zmianie (useEffect z zależnością [tasks])
14. Wczytuj zadania z localStorage przy starcie aplikacji (lazy initial state)
15. Dodaj przycisk "Wyczyść wszystko" z potwierdzeniem

### Część B - Pobieranie danych z API (12 pkt)

8. Stwórz komponent QuoteOfTheDay.jsx pobierający cytat z darmowego API (np. api.quotable.io/random)
9. Wyświetl cytat i autora, dodaj przycisk "Nowy cytat"
10. Obsłuż stany: loading (spinner/"Ładowanie..."), error (komunikat z przyciskiem "Spróbuj ponownie")
11. Dodaj komponent do nagłówka aplikacji

### Część C - Mockowe API dla zadań (10 pkt)

10. Stwórz plik api/tasksApi.js z funkcjami symulującymi API (fetchTasks, saveTasks) z opóźnieniem
11. Użyj tych funkcji w App.jsx: pobierz zadania przy montowaniu, zapisuj przy zmianie
12. Dodaj globalny stan ładowania: skeleton loader podczas pobierania, "Zapisywanie..." podczas zapisu
13. Obsłuż cleanup w useEffect (AbortController)

## Kryteria oceny

- Zapis do localStorage (3 pkt)
- Odczyt z localStorage przy starcie (3 pkt)
- Lazy initial state (2 pkt)
- Pobieranie cytatu z API (4 pkt)
- Obsługa loading state (3 pkt)
- Obsługa error state (3 pkt)
- Mockowe API dla zadań (4 pkt)
- Globalny stan ładowania (3 pkt)
- Cleanup function (2 pkt)
- Obsługa błędów i UX (3 pkt)

## Wskazówki

- useEffect z pustą tablicą [] wykonuje się raz przy montowaniu
- useEffect z zależnościami [tasks] wykonuje się przy każdej zmianie tasks
- Lazy initial state: useState(() => ekspensywnaFunkcja()) - funkcja wykona się tylko raz
- Cleanup: useEffect(() => { ... return () => cleanup(); }, [])

## Zadanie 6: Projekt końcowy - Kompletna aplikacja

<b>Poziom trudności:</b>	3  (Zaawansowany)
<b>Szacowany czas:</b>	60 minut
<b>Punktacja:</b>	20 punktów
<b>Typ:</b>	Syntetyczne/Twórcze
<b>Powiązanie:</b>	Rozszerza wszystkie poprzednie zadania - finalizacja projektu

## Cel dydaktyczny

Połączenie wszystkich poznanych koncepcji React w kompletną aplikację z rozbudowaną funkcjonalnością.

## Polecenie

Rozbuduj Menedżer Zadań o kategorie oraz funkcje wyszukiwania i sortowania:

### Część A - Kategorie zadań (10 pkt)

16. Dodaj pole kategorii do zadań: predefiniowane kategorie ("Praca", "Dom", "Zakupy", "Inne"), select w formularzu, kolorowy badge przy zadaniu
17. Rozszerz model zadania o pole category
18. Dodaj filtrowanie po kategoriach: dropdown/przyciski do wyboru, opcja "Wszystkie kategorie", filtr działa razem z filtrem statusu

### Część B - Wyszukiwanie i sortowanie (10 pkt)

12. Dodaj pole wyszukiwania: input tekstowy nad listą, filtry po tytule (case-insensitive), ikona lupy lub placeholder "Szukaj..."
13. Dodaj sortowanie: dropdown z opcjami "Domyślnie", "Priorytet", "Alfabetycznie", sortowanie działa na przefiltrowanej liście
14. Wyświetl informację gdy brak wyników: "Nie znaleziono zadań dla frazy «...»"

## Kryteria oceny

- System kategorii z badge'ami (4 pkt)
- Filtrowanie po kategoriach (4 pkt)
- Łączenie filtrów (kategoria + status) (2 pkt)
- Wyszukiwanie po tytule (4 pkt)
- Sortowanie z dropdown (4 pkt)
- Obsługa braku wyników (2 pkt)

## Wskazówki

Filtrowanie łączone - zastosuj filtry sekwencyjnie:

 **Wskazówka:** Użyj `.filter().filter().filter().sort()` do łączenia filtrów kategorii, statusu i wyszukiwania, a następnie sortowania.

- Sortowanie po priorytecie: stwórz mapę `{ high: 3, medium: 2, low: 1 }` i sortuj po wartościach
- Wyszukiwanie case-insensitive: użyj `.toLowerCase()` po obu stronach porównania
- Badge kategorii: prosty `<span>` z odpowiednią klasą CSS dla każdej kategorii

## Dodatkowe Informacje

### Sugerowana kolejność wykonywania

Zadania są zaprojektowane sekwencyjnie - każde buduje na poprzednim:

**Zadanie 1 → Zadanie 2 → Zadanie 3 → Zadanie 4 → Zadanie 5 → Zadanie 6**

Każde zadanie rozszerza tę samą aplikację "Menedżer Zadań", dodając nowe funkcjonalności.

### Opcjonalne zadania rozszerzające

- Dodaj Context API do zarządzania globalnym stanem
- Zaimplementuj React Router dla wielu widoków
- Dodaj testy jednostkowe (Jest + React Testing Library)
- Stwórz wersję TypeScript aplikacji
- Zintegruj z prawdziwym backendem (Firebase, Supabase)
- Dodaj drag & drop do zmiany kolejności zadań
- Zaimplementuj dark mode z użyciem CSS variables

## Pomocne zasoby

### Dokumentacja i kursy

- React Documentation: [react.dev](https://react.dev)
- React Tutorial: [react.dev/learn](https://react.dev/learn)
- MDN - JavaScript ES6+: [developer.mozilla.org](https://developer.mozilla.org)

### Narzędzia

- Vite: [vitejs.dev](https://vitejs.dev)
- React DevTools (rozszerzenie przeglądarki)
- VS Code + ES7+ React/Redux/React-Native snippets

### Biblioteki pomocnicze

- React Icons: [react-icons.github.io/react-icons](https://react-icons.github.io/react-icons)
- clsx (łączenie klas CSS): [npmjs.com/package/clsx](https://npmjs.com/package/clsx)
- date-fns (formatowanie dat): [date-fns.org](https://date-fns.org)

### Darmowe API do ćwiczeń

- JSONPlaceholder: [jsonplaceholder.typicode.com](https://jsonplaceholder.typicode.com)
- Quotable (cytaty): [github.com/lukePeavey/quotable](https://github.com/lukePeavey/quotable)
- Public APIs lista: [github.com/public-apis/public-apis](https://github.com/public-apis/public-apis)

## Legenda poziomów trudności

Poziom	Emoji	Nazwa	Opis
1	🟢	Podstawowy	Sprawdzenie zrozumienia podstawowych koncepcji
2	🟡	Średniozaawansowany	Zastosowanie wiedzy w typowych sytuacjach
3	🟠	Zaawansowany	Analiza, synteza, rozwiązywanie złożonych problemów

4		Ekspert	Tworzenie nowej wiedzy, innowacyjne podejścia
---	---	---------	---