

SPRAWOZDANIE Z LABORATORIUM

Technologie Webowe (React)

INFORMACJE PODSTAWOWE

Imię i nazwisko studenta: Patryk Broński

Numer albumu: 27598

Grupa: inf26+ gr.3 spec.programowanie

Data wykonania laboratorium: 17.01.2026

Numer laboratorium: 4

Temat laboratorium: Podstawy React JS

1. CEL LABORATORIUM

Celem laboratorium było zapoznanie się z podstawami biblioteki React oraz praktyczne zastosowanie jej mechanizmów w postaci kompletnej aplikacji typu „Menedżer Zadań”. W ramach ćwiczenia należało nauczyć się tworzenia komponentów funkcyjnych, korzystania ze składni JSX, zarządzania stanem aplikacji za pomocą hooków useState i useEffect, przekazywania danych przez props, obsługi formularzy oraz pracy z localStorage i mockowym API. Dodatkowym celem było uporządkowanie projektu w logiczną strukturę plików oraz rozdzielenie odpowiedzialności pomiędzy komponenty.

2. WYKONANE ZADANIA

Zadanie 1: Pierwszy komponent React

Status: **Wykonane w pełni** Wykonane częściowo Nie wykonane

Opis realizacji:

Utworzyłem podstawową strukturę aplikacji „Menedżer Zadań” w React. Zbudowałem pierwsze komponenty funkcyjne w folderze src/components, w tym komponent nagłówka Header.jsx, oraz połączylem je w pliku App.jsx. Konfigurację uruchomienia aplikacji zrealizowałem w pliku main.jsx, gdzie renderowany jest komponent App przy użyciu React DOM.

Napotkane problemy:

Brak.

Rozwiązanie:

Nie dotyczy.

Zadanie 2: Props - przekazywanie danych

Status: **Wykonane w pełni** Wykonane częściowo Nie wykonane

Opis realizacji:

Zaimplementowałem przekazywanie danych pomiędzy komponentami za pomocą propsów. W komponencie TaskList.jsx przekazałem tablicę zadań i wyrenderowałem elementy listy przy użyciu metody .map(), stosując unikalne klucze key={id}. W komponencie TaskItem.jsx odebrałem dane takie jak tytuł zadania, status wykonania, priorytet oraz kategoria, a także funkcje obsługujące akcje użytkownika. Dodatkowo utworzyłem komponent Card.jsx, który wykorzystałem do kompozycji interfejsu przy użyciu props.children.

Napotkane problemy:

Początkowo miałem trudność z uporządkowaniem dużej liczby propsów przekazywanych do komponentów.

Rozwiązanie:

Zastosowałem destrukturyzację propsów i konsekwentne przekazywanie tylko niezbędnych danych.

Zadanie 3: State i obsługa zdarzeń

Status: **Wykonane w pełni** Wykonane częściowo Nie wykonane

Opis realizacji:

Przeniosłem listę zadań do stanu aplikacji w pliku App.jsx, wykorzystując hook useState. Zaimplementowałem funkcje obsługujące zdarzenia użytkownika, takie jak dodawanie zadania, usuwanie, oznaczanie jako wykonane oraz zmiana priorytetu i kategorii. Dodałem również filtrowanie zadań według statusu, kategorii oraz wyszukiwania, a także sortowanie listy. Statystyki zadań obliczyłem dynamicznie w komponencie TaskStats.jsx na podstawie aktualnego stanu.

Napotkane problemy:

Największym wyzwaniem było połączenie kilku filtrów jednocześnie bez mutowania stanu.

Rozwiązanie:

Użyłem metod `.filter()` i `.map()` na kopii tablicy zadań oraz zadbałem o niemutowalność stanu.

Zadanie 4: Formularze w React

Status: **Wykonane w pełni** Wykonane częściowo Nie wykonane

Opis realizacji:

Utworzyłem kontrolowany formularz dodawania zadań w komponentie `TaskForm.jsx`. Zastosowałem osobny stan dla pól formularza oraz dodałem walidację tytułu zadania (minimalna i maksymalna długość). Zablokowałem możliwość dodania zadania, gdy formularz jest niepoprawny. W komponentie `TaskItem.jsx` zaimplementowałem tryb edycji zadania z obsługą klawiatury, gdzie klawisz Enter zapisuje zmiany, a Escape anuluje edycję.

Napotkane problemy:

Synchronizacja edytowanej wartości z aktualnym tytułem zadania.

Rozwiązanie:

Zastosowałem hook `useEffect` do aktualizacji lokalnego stanu edycji po zmianie propsów.

Zadanie 5: useEffect i pobieranie danych

Status: **Wykonane w pełni** Wykonane częściowo Nie wykonane

Opis realizacji:

Wykorzystałem hook `useEffect` do obsługi efektów ubocznych w aplikacji. Zaimplementowałem wczytywanie zadań z `localStorage` przy starcie aplikacji oraz zapisywanie ich przy każdej zmianie stanu. Dodatkowo użyłem mockowego API w pliku `tasksApi.js`, które symuluje opóźnienie zapytań. W komponentie `QuoteOfTheDay.jsx` pobierałem dane z zewnętrznego API, obsługując stany ładowania oraz błędy.

Napotkane problemy:

Obsługa przerwania zapytania sieciowego podczas odmontowania komponentu.

Rozwiązanie:

Użyłem `AbortController` i funkcji `cleanup` w `useEffect`.

Zadanie 6: Projekt końcowy – Kompletna aplikacja

Status: **Wykonane w pełni** Wykonane częściowo Nie wykonane

Opis realizacji:

Zintegrowałem wszystkie wcześniej wykonane elementy w jedną, spójną aplikację. Dodałem obsługę kategorii zadań, wyszukiwanie, sortowanie oraz filtrowanie działające jednocześnie. Zaimplementowałem przycisk czyszczenia wszystkich zadań z potwierdzeniem oraz zadbałem o spójne stylowanie interfejsu w pliku App.css. Finalnie aplikacja spełnia wszystkie założenia projektu „Menedżer Zadań”.

Napotkane problemy:

Dopasowanie układu interfejsu przy dużej liczbie kontrolek.

Rozwiązanie:

Uporządkowałem layout przy użyciu siatki CSS i logicznego podziału komponentów.

3. FRAGMENTY KODU

Src/App.jsx
src/App.css
src/main.jsx
src/index.css
Src/api/tasksApi.js
Src/components/Card.jsx
Src/components/CategoryFilter.jsx
Src/components/FilterButtons.jsx
Src/components/Header.jsx
Src/components/QuoteOfTheDay.jsx
Src/components/TaskForm.jsx
Src/components/TaskItem.jsx
Src/components/TaskList.jsx
Src/components/TaskStats.jsx

```
import { useEffect, useMemo, useState } from 'react'
import Header from './components/Header'
import Card from './components/Card'
import TaskForm from './components/TaskForm'
import TaskList from './components/TaskList'
import FilterButtons from './components/FilterButtons'
import TaskStats from './components/TaskStats'
import QuoteOfTheDay from './components/QuoteOfTheDay'
import CategoryFilter from './components/CategoryFilter'
import { fetchTasks, saveTasks } from './api/tasksApi'

const STORAGE_KEY = 'task_manager_tasks_v1'

function App() {
  const [tasks, setTasks] = useState(() => {
    try {
      const raw = localStorage.getItem(STORAGE_KEY)
      if (!raw) return []
      const parsed = JSON.parse(raw)
      return Array.isArray(parsed) ? parsed : []
    } catch {
      return []
    }
  })
  const [filter, setFilter] = useState('all')
  const [categoryFilter, setCategoryFilter] = useState('all')
  const [search, setSearch] = useState('')
  const [sortMode, setSortMode] = useState('default')

  const [loading, setLoading] = useState(true)
  const [saving, setSaving] = useState(false)
  const [loadError, setLoadError] = useState('')

  useEffect(() => {
    const controller = new AbortController()
    setLoading(true)
    setLoadError('')
    fetchTasks({ signal: controller.signal })
      .then(data => {
        if (Array.isArray(data)) {
          setTasks(data)
        }
      })
  }, [])
}
```

```

        setLoading(false)
    })
    .catch(e => {
        if (e?.name === 'AbortError') return
        setLoadError('Nie udało się pobrać zadań.')
        setLoading(false)
    })
}

return () => controller.abort()
}, [])

```

```

useEffect(() => {
    try {
        localStorage.setItem(STORAGE_KEY, JSON.stringify(tasks))
    } catch {
        // ignore
    }
}, [tasks])

```

```

useEffect(() => {
    let cancelled = false
    setSaving(true)

    saveTasks(tasks)
        .then(() => {
            if (!cancelled) setSaving(false)
        })
        .catch(() => {
            if (!cancelled) setSaving(false)
        })
}

return () => {
    cancelled = true
}
}, [tasks])

```

```

function addTask(newTask) {
    setTasks(prev => [newTask, ...prev])
}

function toggleTask(id) {
    setTasks(prev => prev.map(t => (t.id === id ? { ...t, completed: !t.completed } : t)))
}

```

```
function deleteTask(id) {
    setTasks(prev => prev.filter(t => t.id !== id))
}

function changePriority(id, newPriority) {
    setTasks(prev => prev.map(t => (t.id === id ? { ...t, priority: newPriority } : t)))
}

function updateTask(id, newTitle) {
    setTasks(prev => prev.map(t => (t.id === id ? { ...t, title: newTitle } : t)))
}

function updateCategory(id, newCategory) {
    setTasks(prev => prev.map(t => (t.id === id ? { ...t, category: newCategory } : t)))
}

function clearAll() {
    const ok = window.confirm('Na pewno usunąć wszystkie zadania?')
    if (!ok) return
    setTasks([])
}

function retryLoad() {
    const controller = new AbortController()
    setLoading(true)
    setLoadError("")

    fetchTasks({ signal: controller.signal })
        .then(data => {
            if (Array.isArray(data)) setTasks(data)
            setLoading(false)
        })
        .catch(e => {
            if (e?.name === 'AbortError') return
            setLoadError('Nie udało się pobrać zadań.')
            setLoading(false)
        })
}

const visibleTasks = useMemo(() => {
    const byStatus = tasks.filter(t => {
        if (filter === 'completed') return t.completed
        if (filter === 'active') return !t.completed
        return true
    })
})
```

```

        })

const byCategory = byStatus.filter(t => {
  if (categoryFilter === 'all') return true
  return (t.category || 'Inne') === categoryFilter
})

const bySearch = byCategory.filter(t =>
  t.title.toLowerCase().includes(search.trim().toLowerCase()))

const sorted = [...bySearch]

if (sortMode === 'alpha') {
  sorted.sort((a, b) => a.title.localeCompare(b.title))
} else if (sortMode === 'priority') {
  const w = { high: 3, medium: 2, low: 1 }
  sorted.sort((a, b) => (w[b.priority] || 0) - (w[a.priority] || 0))
}

return sorted
}, [tasks, filter, categoryFilter, search, sortMode])

const noResults = !loading && tasks.length > 0 && visibleTasks.length === 0

return (
  <div className='app'>
    <Header saving={saving} />

    <div className='topRow'>
      <QuoteOfTheDay />
    </div>

    <Card title='Dodaj zadanie'>
      <TaskForm addTask={addTask} />
    </Card>

    <Card title='Sterowanie'>
      <div className='controls'>
        <FilterButtons filter={filter} setFilter={setFilter} />
      </div>
      <CategoryFilter value={categoryFilter} onChange={setCategoryFilter} />
      <div className='searchBox'>

```

```
<input className='input' value={search} onChange={e =>
setSearch(e.target.value)} placeholder='Szukaj...' />
</div>

<div className='sortBox'>
  <select className='select' value={sortMode} onChange={e =>
setSortMode(e.target.value)}>
    <option value='default'>Domyslnie</option>
    <option value='priority'>Priorytet</option>
    <option value='alpha'>Alfabetycznie</option>
  </select>
</div>

<button className='btn danger' onClick={clearAll}>
  Wyczysc wszystko
</button>
</div>

<TaskStats tasks={tasks} />
</Card>

<Card title='Lista zadań'>
  {loading && (
    <div className='skeleton'>
      <div className='line' />
      <div className='line' />
      <div className='line' />
    </div>
  )}
  {!loading && loadError && (
    <div className='errorBox'>
      <p>{loadError}</p>
      <button className='btn' onClick={retryLoad}>
        Spróbuj ponownie
      </button>
    </div>
  )}
  {!loading && !loadError && (
    <>
      {noResults ? (
        <p className='muted'>Nie znaleziono zadań dla frazy «{search.trim()}»</p>
      ) : (

```

```
<TaskList
  tasks={visibleTasks}
  toggleTask={toggleTask}
  deleteTask={deleteTask}
  changePriority={changePriority}
  updateTask={updateTask}
  updateCategory={updateCategory}
/>
)}
```

```
</Card>
</div>
)
}

export default App
```

4. ZRZUTY EKRANU

The screenshot shows a web application running on localhost:5173. The interface is divided into several sections:

- Menedżer Zadań**: A quote by Anonymous: "Knowledge is being aware of what you can do. Wisdom is knowing when not to do it." with a link to "Nowy cytat".
- Dodaj zadanie**: Form fields for Title (Np. Zrobić zadanie z Reacta), Priority (medium), Category (Inne), and a "Dodaj zadanie" button.
- Sterowanie**: Filter controls for All, Active, Completed, and Categories, along with search and default buttons, and a "Wyczyść wszystko" button. Status summary: Łącznie: 2 Ukończone: 1 Pozostałe: 1 Procent: 50%.
- Lista zadań**: Task list with two items: "posprzątać" (checkbox checked, priority low, category Dom) and "zrobic zadanie z reacta" (checkbox checked, priority low, category Dom).

5. WNIOSKI I REFLEKSJE

Podczas laboratorium nauczyłem się praktycznego tworzenia aplikacji w React z wykorzystaniem komponentów funkcyjnych i składni JSX. Poznałem sposób zarządzania stanem aplikacji przy użyciu hooków useState i useEffect, przekazywania danych pomiędzy komponentami za pomocą propsów oraz obsługi formularzy kontrolowanych. Dodatkowo

nauczyłem się zapisywania i odczytu danych z localStorage oraz symulowania pracy z API przy użyciu mockowych funkcji.

Najtrudniejsze było poprawne zarządzanie stanem aplikacji przy jednoczesnym łączeniu kilku filtrów (status, kategoria, wyszukiwanie i sortowanie) bez bezpośredniej modyfikacji danych. Wymagało to zrozumienia niemutowalności stanu oraz odpowiedniego uporządkowania logiki filtrowania i renderowania komponentów.

W przyszłości chciałbym bardziej przećwiczyć zarządzanie stanem globalnym przy użyciu Context API oraz integrację aplikacji React z prawdziwym backendem. Chciałbym również lepiej poznać optymalizację wydajności aplikacji oraz pisanie testów komponentów w React.

6. SAMOOCENA

Ocena własnego zaangażowania: Bardzo wysokie **Wysokie** Średnie Niskie

Procent wykonanych zadań: 100 %

Dodatkowe uwagi: brak

Data wypełnienia sprawozdania: 31.01.2026

Podpis studenta: Patryk Broński