

Создай React-компонент KanbanBoard, который получает из RTK Query структуру { columns, tasks }, рендерит колонки со списками карточек и поддерживает перенос карточек между колонками через HTML5 Drag&Drop. Добавь обработчик onDrop для смены колонки.

Ответ

Реализуем компонент KanbanBoard с использованием React, Redux Toolkit Query (RTK Query) и HTML5 Drag & Drop API. Компонент будет получать список колонок (columns) и задачи (tasks), визуализируя карточки и поддерживая перетаскивание элементов.

Шаги реализации:

Получаем структуры данных columns и tasks. Рендерим каждую колонку как отдельную секцию. Карточки внутри каждой секции будут поддерживать Drag & Drop поведение. Обработчики события позволят менять позицию карточек между колонками.

Код компонента

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { useGetDataQuery } from './apiSlice'; // Предположим, мы используем RTK Query для фетчинга данных

// Типовая структура компонентов KanbanCard и ColumnWrapper
const KanbanCard = ({ task }) => {
  return (
    <div className="kanban-card">
      <h3>{task.title}</h3>
      <p>{task.description}</p>
    </div>
  );
};

const ColumnWrapper = ({ columnId, title, children }) => {
  const dispatch = useDispatch();

  const handleDragStart = (event, taskId) => {
    event.dataTransfer.setData('taskId', taskId);
  };

  const handleDrop = (event) => {
    const taskId = event.dataTransfer.getData('taskId');
```

```

    dispatch(moveTask({ taskId, newColumnId: columnId }));
  };

  return (
    <div
      className="column-wrapper"
      onDrop={handleDrop}
      onDragOver={(e) => e.preventDefault()}
    >
      <h2>{title}</h2>
      {children}
    </div>
  );
};

export default function KanbanBoard() {
  const { data, isLoading, error } = useGetDataQuery(); // Данные загружаются через
RTK Query
  if (isLoading) return <div>Загрузка...</div>;
  if (error) return <div>Ошибка загрузки данных.</div>;

  const { columns, tasks } = data || {};

  return (
    <div className="kanban-board">
      {Object.keys(columns).map((columnId) => {
        const columnTitle = columns[columnId].title;
        const filteredTasks = Object.values(tasks)
          .filter(task => task.column === columnId); // Фильтруем задачи,
относящиеся к конкретной колонке

        return (
          <ColumnWrapper key={columnId} columnId={columnId} title={columnTitle}>
            {filteredTasks.map((task) => (
              <div
                draggable
                key={task.id}
                onDragStart={(event) => handleDragStart(event, task.id)}
              >

```

```

        <KanbanCard task={task} />
      </div>
    )))
  </ColumnWrapper>
);
}}}
</div>
);
}

```

Что важно учесть:

1. **Drag & Drop:** Реализация основана на стандартных событиях браузера `dragstart`, `dragover`, `drop`. Это обеспечивает базовую поддержку перемещения элементов.
2. **Обработка изменения состояния:** Мы добавили обработчик `moveTask()` в качестве примера, который отправляет новое состояние задачи через редьюсер. Его реализация должна быть предусмотрена в вашем слое управления состоянием приложения.
3. **Стилизация:** Стили CSS не включены сюда, однако они необходимы для оформления интерфейса (например, ширина колонок, стили карт).

Заключение

Мы создали функциональный компонент `KanbanBoard`, способный отображать динамические списки карточек с поддержкой перетаскивания между колонками. Если нужны дополнительные улучшения (например, анимация, расширенная логика сортировки или работа с сервером при изменении позиций), этот код легко масштабируется путем добавления соответствующих функций и изменений стилей.

Доработать текст

Дополни компонент использованием `createApi` из Redux Toolkit Query: запрос `getBoard` (GET `/api/v1/board`) и мутация `reorder` (POST `/api/v1/board/reorder`). Реализуй оптимистичное обновление через `onQueryStarted` и `updateQueryData` с откатом при ошибке.

Ответ

Для дополнения компонента функционалом RTK Query с оптимистичным обновлением воспользуемся функцией `createApi` из Redux Toolkit Query, а также создадим запросы для получения доски (`getBoard`) и её обновления (`reorder`).

Полностью переработанный пример включает следующее:

Определение запросов через `createApi: getBoard` и `reorder`. Оптимистическое обновление с откатом на случай ошибок. Поддержка автоматического сброса оптимистичного патча при возникновении проблем.

Step-by-step implementation:

Шаг 1: Настройка `createApi`

Определим `createApi` для нашего сервиса Kanban Board:

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';

export const kanbanApi = createApi({
  reducerPath: 'kanbanApi',
  baseQuery: fetchBaseQuery({ baseUrl: '/api/v1' }),
  endpoints: builder => ({
    getBoard: builder.query({
      query: () => `/board`,
    }),
    reorder: builder.mutation({
      query: body => ({
        url: '/board/reorder',
        method: 'POST',
        body,
      }),
    }),
  }),
});

export const { useGetBoardQuery, useReorderMutation } = kanbanApi;
```

Шаг 2: Оптимистичное обновление в компоненте

Используя `onQueryStarted`, выполним оптимистичное обновление сразу после начала запроса и отменим его, если произойдёт ошибка. Для этого нам потребуется добавить новый хук (`useReorderMutation`) и вызывать его с соответствующим действием.

Допустим, наша задача состоит в следующем: перенести карточку из одной колонки в другую путём изменения поля `column` в задаче.

Вот полный вариант компонента с оптимистическим обновлением:

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { useGetBoardQuery, useReorderMutation } from './kanbanApi';
```

```

// Допустим, это ваш ранее созданный компонент Card и Column
const KanbanCard = ({ task }) => {
  return (
    <div className="kanban-card">
      <h3>{task.title}</h3>
      <p>{task.description}</p>
    </div>
  );
};

const ColumnWrapper = ({ columnId, title, children }) => {
  const dispatch = useDispatch();

  const handleDragStart = (event, taskId) => {
    event.dataTransfer.setData('taskId', taskId);
  };

  const handleDrop = async (event) => {
    const taskId = event.dataTransfer.getData('taskId');
    await moveTask(taskId, columnId);
  };

  return (
    <div
      className="column-wrapper"
      onDrop={handleDrop}
      onDragOver={(e) => e.preventDefault()}
    >
      <h2>{title}</h2>
      {children}
    </div>
  );
};

export default function KanbanBoard() {
  const { data, isLoading, error } = useGetBoardQuery();
  const [reorderTask, { isLoading: mutationIsLoading }] = useReorderMutation();

  if (isLoading) return <div>Загрузка...</div>;

```

```

if (error) return <div>Ошибка загрузки данных.</div>;

const { columns, tasks } = data || {};

const moveTask = async (taskId, newColumnId) => {
  try {
    await reorderTask({ taskId, newColumnId });
  } catch (err) {
    console.error("Ошибка при перемещении:", err);
  }
};

return (
  <div className="kanban-board">
    {Object.keys(columns).map((columnId) => {
      const columnTitle = columns[columnId].title;
      const filteredTasks = Object.values(tasks)
        .filter(task => task.column === columnId);

      return (
        <ColumnWrapper key={columnId} columnId={columnId} title={columnTitle}>
          {filteredTasks.map((task) => (
            <div
              draggable
              key={task.id}
              onDragStart={(event) => handleDragStart(event, task.id)}
            >
              <KanbanCard task={task} />
            </div>
          ))}
        </ColumnWrapper>
      );
    })}
  </div>
);
}

```

Step 3: Использование оптимистического обновления
 через `updateQueryData`

Чтобы реализовать оптимистичное обновление, нужно изменить компонент следующим образом:

```
// Определяем endpoint в канбанAPI.js с поддержкой оптимистичных обновлений
reorder: builder.mutation({
  query: body => ({
    url: '/board/reorder',
    method: 'POST',
    body,
  }),
  onQueryStarted: async ({ taskId, newColumnId }, { dispatch, queryFulfilled,
getState }) => {
    const patchResult = dispatch(
      kanbanApi.util.updateQueryData('getBoard', undefined, draft => {
        const updatedTaskIndex = draft.tasks.findIndex(t => t.id === taskId);
        if (updatedTaskIndex !== -1) {
          draft.tasks[updatedTaskIndex].column = newColumnId;
        }
      })
    );

    try {
      await queryFulfilled;
    } catch (err) {
      patchResult.undo();
    }
  },
}),
```

Теперь каждый раз, когда начинается операция перемещения карточки, мы немедленно применяем локальное изменение данных (обновляем столбец задачи), а если произойдет ошибка, восстановление выполняется автоматически.

ИТОГОВЫЙ ВЫВОД:

Мы реализовали полноценный компонент Kanban Board с оптимистическими изменениями через RTK Query. Теперь операции перетаскивания мгновенно отражают ожидаемые изменения, а любые возникшие проблемы приводят к автоматическому откату изменений, сохраняя консистентность данных.

Этот подход улучшает взаимодействие пользователей с приложением, делая UI отзывчивым и приятным в работе даже при наличии задержек на стороне сервера.

Доработать текст

Добавь ARIA роли: `role='list'` для колонок, `role='listitem'` и `aria-grabbed` для карточек. Добавь фокусируемость карточек `tabIndex={0}` и стиль фокуса. Для колонок добавь визуальную подсветку при `dragover`.

Ответ

Для доступности вашего компонента, важно правильно настроить семантику и интерактивность с точки зрения взаимодействия с клавиатурой и вспомогательными технологиями (screen readers). Давайте рассмотрим доработку вашего компонента с добавлением нужных атрибутов и улучшений.

Улучшения, которые мы внесём:

ARIA роли: Колонки получают роль `role="list"` для обозначения списка задач. Карточки получают роль `role="listitem"` для каждого элемента списка. Атрибут `aria-grabbed` позволит экранному читателю сообщать о состоянии захвата карты. **Фокусировка:** Каждая карта получит атрибут `tabIndex={0}`, чтобы обеспечить возможность навигации с клавиатуры. Будет настроен эффект выделения при фокусировке. **Подсветка колонок:** Во время процесса перетаскивания колонки будут подсвечиваться, чтобы визуально обозначить область куда элемент можно отпустить.

Новый компонент с доступностью и подсветкой

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { useGetBoardQuery, useReorderMutation } from './kanbanApi';

// Компоненты для отображения карточек и колонок
const KanbanCard = ({ task, focused, dragging }) => {
  return (
    <div
      className={`kanban-card ${focused ? 'focussed' : ''}`}
      style={{ backgroundColor: dragging && 'lightyellow' }}
      role="listitem"
      aria-grabbed={dragging ? true : false}
      tabIndex={0}
    >
      <h3>{task.title}</h3>
      <p>{task.description}</p>
    </div>
  );
};
```



```

const ColumnWrapper = ({ columnId, title, children, isDraggingOver }) => {
  return (
    <div
      className="column-wrapper"
      role="list"
      style={{
        border: isDraggingOver ? '2px dashed blue' : '',
        padding: '8px'
      }}
    >
      <h2>{title}</h2>
      {children}
    </div>
  );
};

export default function KanbanBoard() {
  const { data, isLoading, error } = useGetBoardQuery();
  const [reorderTask, { isLoading: mutationIsLoading }] = useReorderMutation();

  if (isLoading) return <div>Загрузка...</div>;
  if (error) return <div>Ошибка загрузки данных.</div>;

  const { columns, tasks } = data || {};

  const moveTask = async (taskId, newColumnId) => {
    try {
      await reorderTask({ taskId, newColumnId });
    } catch (err) {
      console.error("Ошибка при перемещении:", err);
    }
  };

  return (
    <div className="kanban-board">
      {Object.keys(columns).map((columnId) => {
        const columnTitle = columns[columnId].title;
        const filteredTasks = Object.values(tasks)
          .filter(task => task.column === columnId);

```

```

    return (
      <ColumnWrapper key={columnId} columnId={columnId} title={columnTitle}>
        {filteredTasks.map((task) => (
          <div
            draggable
            key={task.id}
            onDragStart={(event) => handleDragStart(event, task.id)}
            onFocus={() => console.log(`Карточка с id=${task.id} получила
фокус`)}

            tabIndex={0}
          >
            <KanbanCard task={task} focused={false} dragging={false} />
          </div>
        ))}
      </ColumnWrapper>
    );
  })}
</div>
);
}

```

// Обработчики событий drag/drop теперь используют доступность и реагируют на перемещение мышью и клавиатурой

```

const handleDragStart = (event, taskId) => {
  event.dataTransfer.setData('taskId', taskId);
  document.addEventListener('dragenter', handleDragEnter);
  document.addEventListener('dragleave', handleDragLeave);
};

const handleDragEnter = (event) => {
  const targetElement = event.target.closest('.column-wrapper');
  if (!targetElement) return;
  targetElement.classList.add('drag-over');
};

const handleDragLeave = (event) => {
  const targetElement = event.target.closest('.column-wrapper');
  if (!targetElement) return;

```

```

targetElement.classList.remove('drag-over');
};

const handleDrop = async (event) => {
  const taskId = event.dataTransfer.getData('taskId');
  const columnId = event.target.closest('.column-wrapper').dataset.columnid;
  await moveTask(taskId, columnId);

  document.removeEventListener('dragenter', handleDragEnter);
  document.removeEventListener('dragleave', handleDragLeave);
};

```

Объяснение изменений:

1. **Колонка** (ColumnWrapper):

- Роль "list" позволяет браузеру понимать, что каждая колонка является списком задач.
- Подсветка при событии dragover (через класс .drag-over) показывает зону, куда можно отпустить элемент.

2. **Карточка** (KanbanCard):

- Добавлена роль "listitem", показывая, что каждая карточка представляет собой отдельный пункт списка.
- Установлен атрибут tabIndex={0}, позволяющий пользователю выбирать карту с помощью клавиши Tab.
- Атрибут aria-grabbed используется для информирования о захвате карточки.

3. **CSS стили для фокуса и подсветки:** Вы можете дополнительно создать стили для эффекта выделения при фокусировании или подсветке области drag-and-drop:

```

.kanban-card.focussed {
  outline: 2px solid #ffa500; /* Оранжевый контур */
}

.drag-over {
  box-shadow: inset 0 0 10px rgba(0, 0, 255, 0.5); /* Синяя внутренняя тень */
}

```

Таким образом, ваш компонент стал более доступным и удобным для всех пользователей, включая тех, кто пользуется клавиатурой или вспомогательной технологией чтения экрана.

Доработать текст

Вынеси карточку в отдельный компонент TaskCard (React.memo). Обеспечь корректные ключи элементов, обработку пустых списков и устойчивость к отсутствию данных.

Ответ

Давайте перенесём карточку в отдельный компонент, используя React.memo для оптимизации производительности. Также добавим проверку на отсутствие данных и корректные ключи элементов.

Разделение компонента на отдельные части

Сначала вынесем функциональность отдельной карточки в отдельный компонент TaskCard. Этот компонент будет использоваться для рендера индивидуальных задач и обрабатываться как мемоизованный элемент, что предотвратит ненужные повторные рендеры.

Затем улучшим основную часть компонента KanbanBoard, добавив проверки отсутствия данных и обработки пустых списков.

Изменённый компонент TaskCard

Здесь приведён оптимальный подход с использованием React.memo:

```
import React from 'react';

const TaskCard = React.memo(({ task, focused, dragging }) => {
  return (
    <div
      className={`kanban-card ${focused ? 'focussed' : ''}`}
      style={{ backgroundColor: dragging && 'lightyellow' }}
      role="listitem"
      aria-grabbed={dragging ? true : false}
      tabIndex={0}
    >
      <h3>{task?.title ?? 'Нет названия'}</h3>
      <p>{task?.description ?? 'Описание отсутствует'}</p>
    </div>
  );
});

export default TaskCard;
```

Основная часть компонента KanbanBoard

Теперь переопределим основной компонент, учитывая новые компоненты и рекомендации по устойчивости:

```
import React from 'react';
```

```

import { useSelector, useDispatch } from 'react-redux';
import { useGetBoardQuery, useReorderMutation } from './kanbanApi';
import TaskCard from './TaskCard';

// Обертка для колонки
const ColumnWrapper = ({ columnId, title, children, isDraggingOver }) => {
  return (
    <div
      className="column-wrapper"
      role="list"
      style={{
        border: isDraggingOver ? '2px dashed blue' : '',
        padding: '8px'
      }}
    >
      <h2>{title}</h2>
      {children.length > 0 ? children : <span>Список пуст</span>}
    </div>
  );
};

export default function KanbanBoard() {
  const { data, isLoading, error } = useGetBoardQuery();
  const [reorderTask, { isLoading: mutationIsLoading }] = useReorderMutation();

  if (isLoading) return <div>Загрузка...</div>;
  if (error) return <div>Ошибка загрузки данных.</div>;

  const { columns, tasks } = data || {};

  const moveTask = async (taskId, newColumnId) => {
    try {
      await reorderTask({ taskId, newColumnId });
    } catch (err) {
      console.error("Ошибка при перемещении:", err);
    }
  };

  return (

```

```

<div className="kanban-board">
  {Object.keys(columns)?.length > 0 ? (
    Object.keys(columns).map(columnId => {
      const columnTitle = columns[columnId]?.title;
      const filteredTasks = Object.values(tasks)
        ?.filter(task => task.column === columnId);

      return (
        <ColumnWrapper key={columnId} columnId={columnId} title={columnTitle}>
          {filteredTasks?.map((task) => (
            <div
              draggable
              key={task.id}
              onDragStart={(event) => handleDragStart(event, task.id)}
              onFocus={() => console.log(`Карточка с id=${task.id} получила
фокус`)}

              tabIndex={0}
            >
              <TaskCard task={task} focused={false} dragging={false} />
            </div>
          ))}
        </ColumnWrapper>
      );
    })
  ) : (
    <div>Нет колонок для отображения</div>
  )}
</div>
);
}

// Обработчики drag-n-drop остаются прежними
const handleDragStart = (event, taskId) => {
  event.dataTransfer.setData('taskId', taskId);
  document.addEventListener('dragenter', handleDragEnter);
  document.addEventListener('dragleave', handleDragLeave);
};

const handleDragEnter = (event) => {

```

```

const targetElement = event.target.closest('.column-wrapper');
if (!targetElement) return;
targetElement.classList.add('drag-over');
};

const handleDragLeave = (event) => {
  const targetElement = event.target.closest('.column-wrapper');
  if (!targetElement) return;
  targetElement.classList.remove('drag-over');
};

const handleDrop = async (event) => {
  const taskId = event.dataTransfer.getData('taskId');
  const columnId = event.target.closest('.column-wrapper')?.dataset.columnid;
  await moveTask(taskId, columnId);

  document.removeEventListener('dragenter', handleDragEnter);
  document.removeEventListener('dragleave', handleDragLeave);
};

```

Основные моменты решения:

1. **Использование React.memo:** TaskCard обернут в React.memo, что предотвращает лишние перерисовки компонента, если пропсы не меняются.
 2. **Проверка наличия данных:** перед итерациями проверяется наличие объектов (data, columns, tasks), предотвращая возможные ошибки при отсутствии данных.
 3. **Корректные ключи:** каждый элемент массива имеет уникальный ключ (key={task.id}), обеспечивающий правильное отслеживание элементов.
 4. **Отображение сообщений при пустых списках:** для ситуаций, когда колонка или доска пусты, предусмотрены соответствующие сообщения ("нет колонок для отображения", "список пуст").
- Эти меры обеспечивают устойчивую работу компонента, улучшение производительности и удобство восприятия пользователями.