TanStack Table — это современная и высоко настраиваемая библиотека для построения таблиц и гридов в React и других JavaScript-фреймворках. Она позиционирует себя как "headless" (без головного рендера), что означает: библиотека отвечает только за логику построения и управления данными, а внешний вид и разметка полностью зависят от вас.

**1. Headless (без шаблонов)**

- Логика полностью отделена от визуальной части.

- Вы сами определяете, как будет выглядеть строка, ячейка, пагинация и другие элементы.

- Нет готовых компонентов: TanStack Table выдает вам "шаблон" состояния, а вы сами реализуете отображение.

**2. Гибкая структура данных**

- Поддерживает любые вложенности: можно строить простые таблицы, древовидные структуры (Tree Table), группировки, поддержку подтаблиц (SubTable).

- Колонки описываются объектами, в которых можно указывать заголовок (`header`), ключ данных (`accessorKey`), индивидуальные компоненты рендера и многое другое.

**3. Мощная архитектура хуков**

- Управляется через хуки, которые возвращают все необходимое для рендера (строки, ячейки, колонки, состояние пагинации и т.д.).

- Например: `useReactTable` — основной хук для создания таблицы.

**4. Полная поддержка TypeScript**

- Библиотека написана с поддержкой типизации "из коробки".

**Ключевые возможности**

- **Сортировка**: Локальная и серверная, поддержка мультисортировки.

- **Фильтрация**: Кастомные фильтры для любой колонки и всей таблицы.

- **Пагинация**: Локальная/серверная, произвольная логика разбиения по страницам.

- **Группировка**: Группировка данных по одной или нескольким колонкам.

- **Агрегация**: Поддержка подсчёта итогов, сумм, средних, и др.

- **Редактирование**: Встроенная поддержка или полностью кастомные решения для редактируемых ячеек/строк.

- **Перетаскивание колонок**: Встроенно или с помощью расширений.

- **Виртуализация**: Работа с огромными таблицами без просадки по производительности.

- Поддержка аксессоров функции и вложенных объектов.

- **Расширения**: Можно реализовать свои плагины или расширять функциональность через хуки.

**Шпаргалка по TanStack Table**

**1. Основной хук и настройки**

**useReactTable(config)**

- Что делает: Запускает таблицу с вашими настройками.

- По-простому:

  Ты этим “включаешь” таблицу и объясняешь ей, где брать данные и как рисовать колонки.

**data**

- Что делает: Массив твоих данных, например [{id:1, name:'Иван'}].

- По-простому:

  Тут лежат все твои клиенты (или что ты хочешь показать в таблице).

**columns**

- Что делает: Определяет, какие будут столбцы и что в них выводить.

- По-простому:

  Ты тут пишешь — хочу столбец “Имя”, хочу столбец “Email”, и какую инфу брать для каждого.

**getCoreRowModel**

- Что делает: Показывает все строки, как они есть, без сортировки и фильтрации.

- По-простому:

  Просто выводит твои данные, как они заданы.

**getSortedRowModel**

- Что делает: Делает сортировку, если она включена.

- По-простому:

  Помогает отсортировать таблицу по колонке, если кликнуть на заголовок.

**getFilteredRowModel**

- Что делает: Показывает только те строки, что прошли фильтр.

- По-простому:

  Оставляет только то, что ищет пользователь (например, поиск по имени).

**getPaginationRowModel**

- Что делает: Делит строки на страницы.

- По-простому:

  Показывает, например, только первые 10 строк, остальное на других страницах.

**state**

- Что делает: Хранит состояния сортировки, фильтрации, страниц.

- По-простому:

  Здесь хранится, что пользователь включил (например: какая сортировка сейчас).

onSortingChange, onGlobalFilterChange, onPaginationChange

- Что делает: Эти функции запускаются, когда пользователь сортирует, фильтрует или листает страницы.

- По-простому:

  Ты сюда пишешь “что делать, когда юзер что-то поменяет” — менять состояние, получать новые данные и т.п.

**const table = useReactTable({**

data, // Ваши данные

columns, // Ваши столбцы

getCoreRowModel: getCoreRowModel(), // Ядро строк

getSortedRowModel: getSortedRowModel(), // Добавляет сортировку

getFilteredRowModel: getFilteredRowModel(), // Добавляет фильтрацию

getPaginationRowModel: getPaginationRowModel(), // Добавляет пагинацию

state: { sorting, globalFilter, pagination }, // Состояния таблицы

onSortingChange: setSorting, // Обработчик сортировки

onGlobalFilterChange: setGlobalFilter, // Обработчик глобального фильтра

onPaginationChange: setPagination, // Обработчик пагинации

});

**2. Методы таблицы**

**getHeaderGroups()**

- Что делает: Достаёт все группы заголовков для таблицы.

- По-простому:

  Достаёт названия твоих колонок, чтобы их красиво нарисовать сверху.

<table>

<thead>

{table.getHeaderGroups().map(headerGroup => (

<tr key={headerGroup.id}>

{headerGroup.headers.map(header => (

<th

key={header.id}

onClick={header.column.getToggleSortingHandler()}

style={{ cursor: 'pointer' }}

>

**getRowModel()**

это основная модель строки, которую следует использовать для рендеринга разметки строк таблицы.

1.строки — массив строк table.getRowModel().rows ().

2.flatRows — массив строк, в котором все вложенные строки объединены в один уровень table.getRowModel().flatRows.

3.rowsById — объект со строками, где каждая строка имеет ключ id. Это полезно для быстрого поиска строк по их id с более высокой производительностью table.getRowModel().rowsById['row-id'].

- Что делает: Достаёт все строки, которые надо показать пользователю по текущим настройкам.

- По-простому:

  Список всех “клиентов”, что сейчас отображаются (фильтрация, пагинация и сортировка уже учтены).

<tbody>

{table.getRowModel().rows.map(row => (

<tr key={row.id}>

{row.getVisibleCells().map(cell => (

<td key={cell.id}>

{flexRender(cell.column.columnDef.cell, cell.getContext())}

</td>

))}

</tr>

))}

</tbody>

**getState()**

- Что делает: Показывает текущее состояние таблицы (сортировка, фильтр, страница).

- По-простому:

  "Записная книжка" — что сейчас включено (например, сортируем по имени или находимся на второй странице).

console.log(table.getState()); // { sorting: [...], globalFilter: '', pagination: {...} }

**getPageCount()**

- \*\*Что делает:\*\* Считает, сколько всего страниц данных сейчас.

- \*\*По-простому:\*\*

  Просто говорит, сколько всего страниц у тебя получится.

let totalPages = table.getPageCount();

console.log(`Всего страниц: ${totalPages}`);

**previousPage()**

- Что делает: Переключает таблицу на предыдущую страницу.

- По-простому:

  Кнопка “назад”.

<button onClick={() => table.previousPage()}>Назад</button>

**nextPage()**

- Что делает: Переключает таблицу на следующую страницу.

- По-простому:

  Кнопка “вперёд”.

<button onClick={() => table.nextPage()}>Вперёд</button>

**getCanPreviousPage()**

- Что делает: Проверяет, можно ли идти на страницу назад.

- По-простому:

  Если уже на первой странице, кнопка “назад” отключится.

<button disabled={!table.getCanPreviousPage()}>Назад</button>

**getCanNextPage()**

- Что делает: Проверяет, можно ли идти на следующую страницу.

- По-простому:

  Если уже на последней странице — кнопка “вперёд” отключится.

<button disabled={!table.getCanNextPage()}>Вперёд</button>

**Методы внутри header и row**

**renderHeader()**

(или flexRender(header.column.columnDef.header, ...))

- Что делает: Отрисовывает содержимое шапки колонки.

- По-простому:

  Рисует текст или кнопки в верхних ячейках столбца.

{flexRender(header.column.columnDef.header, header.getContext())}

**column.getToggleSortingHandler()**

- Что делает: Достаёт функцию, чтобы клик по заголовку запускал сортировку.

- По-простому:

  Позволяет сортировать столбец кликом.

<th onClick={header.column.getToggleSortingHandler()}>

{flexRender(header.column.columnDef.header, header.getContext())}

</th>

**column.getIsSorted()**

- Что делает: Проверяет, отсортирован ли столбец и как.

- По-простому:

  Показывает, есть ли здесь сортировка (и какую стрелочку нарисовать).

const sortType = header.column.getIsSorted(); // 'asc', 'desc' или false

**row**

**getVisibleCells()**

- Что делает: Возвращает все ячейки, которые надо показать для этой строки.

- По-простому:

  Позволяет получить значения во всех столбцах текущей строки.

row.getVisibleCells().map(cell => ...)

**cell**

renderCell()

(или flexRender(cell.column.columnDef.cell, ...))

- Что делает: Отрисовывает содержимое ячейки (в строке таблицы).

- По-простому:

  Просто пишет текст (например, имя или email) в нужной клетке.

{flexRender(cell.column.columnDef.cell, cell.getContext())}

**3. Работа с состояниями**

**sorting**

- Что делает: Хранит, по какому столбцу и как сортировать таблицу.

- По-простому:

  Запоминает выбор пользователя, если он нажал сортировку.

const [sorting, setSorting] = useState([]);

**globalFilter**

- Что делает: Хранит текст поиска для всей таблицы.

- По-простому:

  Сюда сохраняется, что пользователь ввёл в поиск.

const [globalFilter, setGlobalFilter] = useState('');

**pagination**

- Что делает: Хранит номер текущей страницы и сколько строк на странице.

- По-простому:

  Тут написано, на какой сейчас странице таблицы ты находишься.

const [pagination, setPagination] = useState({ pageIndex: 0, pageSize: 10 });

**4. Столбцы / Колонки**

**`header`**

- Что делает: Название/элемент для заголовка столбца.

- По-простому:

  Это то, что будет написано сверху в таблице — например “Имя”.

**accessorKey**

- Что делает: Ключ по которому брать данные из строки.

- По-простому:

  Показывает, откуда брать данные для этой колонки (например, "name" из {name: "Вася"}).

{ accessorKey: 'name', header: 'Имя' }

---

## МИНИМАЛЬНЫЙ НАБОР

| Название | Что делает |

| useReactTable | Запускает таблицу |

| getHeaderGroups | Достаёт названия всех колонок |

| getRowModel | Возвращает все видимые строки |

| getState | Достаёт текущее “состояние” |

| previousPage | Кнопка “назад” для страниц |

| nextPage | Кнопка “вперёд” для страниц |

| getCanPreviousPage | Можно ли идти “назад” по страницам |

| getCanNextPage | Можно ли идти “вперёд” по страницам |

| header, accessorKey | Настройки названия и содержимого колонки

| getVisibleCells | Получить все клетки строки |

| renderCell, renderHeader| Что показывать в заголовках и ячейках |

**Кратко:**

- Всё начинается с useReactTable — туда передаётся структура данных и функций.

- Для отображения используем методы table: getHeaderGroups, getRowModel и др.

- Фильтрация, сортировка, пагинация — через состояния и методы table.

- Внешний вид (HTML, стили, оформление) вы делаете сами!

**ПРИМЕР**

import React, { useMemo, useState } from 'react';

import { useReactTable, getCoreRowModel, getSortedRowModel, getFilteredRowModel, getPaginationRowModel } from '@tanstack/react-table';

// Данные для таблицы

const data = [

{ id: 1, name: 'Иван', email: 'ivan@mail.ru', age: 28 },

{ id: 2, name: 'Мария', email: 'maria@mail.ru', age: 34 },

{ id: 3, name: 'Алексей', email: 'aleksey@mail.ru', age: 22 },

{ id: 4, name: 'Ольга', email: 'olga@mail.ru', age: 31 },

// Добавь больше данных для тестирования пагинации

];

// Определение колонок

const columns = [

{ accessorKey: 'name', header: 'Имя' },

{ accessorKey: 'email', header: 'Email' },

{ accessorKey: 'age', header: 'Возраст' },

];

const App = () => {

const [globalFilter, setGlobalFilter] = useState('');

const [sorting, setSorting] = useState([]);

const [pagination, setPagination] = useState({ pageIndex: 0, pageSize: 2 });

// Создание таблицы

const table = useReactTable({

data,

columns,

getCoreRowModel: getCoreRowModel(),

getSortedRowModel: getSortedRowModel(),

getFilteredRowModel: getFilteredRowModel(),

getPaginationRowModel: getPaginationRowModel(),

state: { sorting, globalFilter, pagination },

onSortingChange: setSorting,

onGlobalFilterChange: setGlobalFilter,

onPaginationChange: setPagination,

});

return (

<div>

{/\* Поле для глобального фильтра \*/}

<input

value={globalFilter}

onChange={(e) => setGlobalFilter(e.target.value)}

placeholder="Поиск по всем столбцам"

/>

<table border={1} cellPadding={8} style={{ borderCollapse: 'collapse', margin: '20px 0' }}>

<thead>

{table.getHeaderGroups().map(headerGroup => (

<tr key={headerGroup.id}>

{headerGroup.headers.map(header => (

<th key={header.id} onClick={header.column.getToggleSortingHandler()} style={{ cursor: 'pointer' }}>

{header.renderHeader()} {header.column.getIsSorted() ? (header.column.getIsSorted() === 'asc' ? '🔼' : '🔽') : ''}

</th>

))}

</tr>

))}

</thead>

<tbody>

{table.getRowModel().rows.map(row => (

<tr key={row.id}>

{row.getVisibleCells().map(cell => (

<td key={cell.id}>

{cell.renderCell()}

</td>

))}

</tr>

))}

</tbody>

</table>

{/\* Пагинация \*/}

<div>

<button onClick={table.previousPage} disabled={!table.getCanPreviousPage()}>Назад</button>

<span>{`Страница ${table.getState().pagination.pageIndex + 1} из ${table.getPageCount()}`}</span>

<button onClick={table.nextPage} disabled={!table.getCanNextPage()}>Вперёд</button>

</div>

</div>

);

};