React Profiler в React Developer Tools (DevTools) — это вкладка, которая позволяет замерять производительность React-компонентов в реальном времени. Она показывает, сколько времени потребовалось на рендеринг каждого компонента и помогает находить узкие места и излишние обновления.

**Открой вкладку DevTools:**

Открой твоё приложение в браузере.

Найди вкладку React среди остальных (обычно рядом с Console и Elements).

**Включи Profiler:**

Перейди во вкладку Profiler в React DevTools.

Убедись, что приложение работает под контролем React (компоненты в дереве React будут отмечены специальной галочкой в DevTools).

Как начать запись профилирования

**Зайди во вкладку Profiler.**

Нажми кнопку "Start recording" (Начать запись) в верхнем левом углу.

Выполни несколько действий в приложении (например, клики на кнопки, ввод текста) или дождись, пока приложение обновит состояние.

Нажми "Stop recording" (Остановить запись), чтобы завершить процесс.

Ты увидишь графическую временную шкалу и список компонентов, обновившихся во время записи.

**Что ты увидишь после записи?**

График вложенности компонентов

Это список всех компонентов React, работающих в твоём приложении. Они представлены в иерархическом порядке (родительские компоненты наверху, дочерние — внизу).

**Цветовые индикаторы**

Компоненты, которые занимают больше времени на рендеринг, подсвечиваются красным. Менее затратные — зелёным.

Чем сильнее "краснота", тем больше внимания нужно уделить этому компоненту, чтобы оптимизировать производительность.

**Детали компонентов**

Когда ты кликаешь на компонент в списке, справа появляется дополнительная информация:

"Render Duration": сколько времени компонент потратил на рендеринг.

"Tree Base Duration": суммарное время для всего дерева компонентов, начиная с этого узла.

"Wasted Rendering": время, потраченное на рендеры, которые не привели к изменениям (избыточные обновления).

**Оптимизация на основе данных Profiler**

Данные, которые ты видишь в Profiler, помогут решить следующие проблемы:

Частые перерисовки компонентов

Если компонент перерисовывается слишком часто при любом изменении состояния, его можно оптимизировать.

Например:

**Используй React.memo() для мемоизации компонентов**.

Проверь, не меняются ли ссылки в пропсах (может, ты каждый раз создаёшь новые объекты/функции).

**Долгие рендеры**

Если компонент тратит слишком много времени на рендеринг (красный цвет), это может быть связано с:

Слишком большим количеством JSX-элементов внутри компонента.

Тяжёлыми вычислениями (нужно их вынести в useMemo или useCallback).

Нецелесообразным использованием списков без ключей (key).

**Избыточные обновления**

Убедись, что твои "child" компоненты не обновляются без необходимости. Проверь, правильно ли сделана передача пропсов или состояния.

**Полезные функции в Profiler**

**"Flame graph" и "Ranked"**

Есть два способа анализа данных:

Flame graph: иерархическое дерево компонентов. Чётко видно, сколько времени потратил каждый компонент.

Ranked view: список компонентов по количеству времени, чтобы сразу понять, что "жрёт" больше всего ресурсов.

**Фильтры**

React Profiler позволяет скрыть не важные компоненты (например, системные или обёртки вроде Context.Provider) и сфокусироваться только на значимых.

**Сравнение рендера**

Если ты записал несколько сессий профилирования, можешь сравнить их и проверить, как твои изменения повлияли на скорость.

**Пример использования на практике**

Предположим, что тебе нужно понять, почему приложение тормозит при вводе текста в поле.

Открой Profiler и начни запись.

Введи текст в поле и останови запись.

Найди компонент, связанный с полем ввода (например, ` или `).

Проверь:

Какое время ушло на рендеринг.

Какие компоненты обновились вместе с полем.

Какие из обновлений кажутся лишними (например, unrelated компоненты, рендер которых не должен происходить при вводе текста).

Если ты видишь, что множество компонентов обновились ненужным образом, можно попробовать:

Использовать React.memo.

Разбить большие компоненты на более мелкие, чтобы локализовать изменения.

Добавить мемоизацию состояния или функций через useCallback и useMemo.

Полезные советы для новичков

Используй Profiler только в dev-режиме — он не будет работать в production-сборке.

Не переживай, если видишь много "красных" компонентов — это не всегда плохо. Главное фокусироваться на узких местах, сильно влияющих на производительность.

Разбирай приложение по частям. Если сразу сложно понять информацию, начни с малого: профилируй одну маленькую секцию или компонент.

Профилирование особенно полезно на больших приложениях со сложной логикой. В простых приложениях результаты могут быть менее заметны.