React Concurrent Mode and Suspense Demo.

This project contains example code demonstrating React's upcoming Concurrent Mode and Suspense features.

Instructions

First, run

```
npm install
```

to install project dependencies.

Start the local development server @ localhost:3000 via

```
npm start
```

Overview

The examples are:

- `ex01`

  Demonstrates how a React application becomes unresponsive as the component tree grows, due to the blocking nature of React's render process.

- `ex02`

  Applies non-blocking rendering - a core Concurrent Mode feature - to `ex02`, allowing the application to stay responsive throughout.

- `ex03`

  Illustrates the problem of components fetching their data dependencies without any mutual coordination, resulting in a broken user experience with a lot of loading states ("spinners") being displayed.

- `ex04`

  Improves `ex03` by implementing data fetching with Suspense, enabling easy orchestration of intentional, user-friendly loading sequences.

Comments on example ex04

In contrast to the 3rd party `useSWR` hook (from the `swr` library) used in example `ex03`, in this example a custom `useFetch` hook, designed to be compatible with the new Suspense mechanism, is used for data fetching (see further comments in `fetch.js`.

The data fetching pattern implemented in this example differs significantly from `ex03`:

- In `ex03`, a component, such as `User`, is rendered first ***before*** data fetching commences. The built-in `useEffect` hook is typically used for managing data fetching (the 3rd party `useSWR` hook utilizes `useEffect` in its implementation) and an effect is run after a component's initial render.

  The React documentation refers to this pattern as *fetch-on-render*.

- In `ex04`, data fetching starts ***before rendering the component***, in the `App` component, the `selectUser` event handler initiates work in the background via a `startTransition` call that runs the following:

  - A custom `prefetch` method (see `fetch.js`), which immediately kicks off data fetching for the selected user.
  - A low-priority state update of the `showProfile` flag.

  Updating `showProfile` to `true` will start to render the user profile view as part of the background work. When the `User` component is rendered, the data for the selected user may already have arrived and thus React will ***transition*** from showing the list of users to the user profile view.

  If the data is not available the component will ***suspend***. Again, all of this occurs in the background until the `User` component's data is ready and a transition to the user profile view is made, a period during which the user sees a pending indicator (or the Suspense fallback if the data fetching takes too long, specified by the transition timeout).

  This pattern of fetching data as early as possible is called *fetch-as-you-render*.

References

React Concurrent Mode documentation

Dan Abramov's "inaugural talk about Concurrent Mode

Data Fetching with Suspense in Relay/GraphQL

Contact

Marc Klefter marc@remotifi.com