Sofascore Frontend Academy Lecture 06,
April, 2023

CSS in JS. SWR. Context. Routing.

Alen Murtić

Sofascore

TABLE OF CONTENTS

Sofascore

01

CSS in JS

Motivation

- Classic usage of CSS via classes isn't really for the components era
    - In big projects, those can become mentally unscalable
    - While it supports code reuse, only most basic stuff is re-used
    - E.g. standard project paddings, margins, borders
    - <u>CSS is focused on defining document-level stylesheets, not component-level</u>
- Solution: CSS in JS

CSS in JS

- Idea: Write CSS inside JS files, leverage some JS features
- Advantages:
  - Thinking in components
  - Inject only used styles at render-time, not all styles
  - Handles vendor prefixing (e.g. -webkit-box-align or -moz-box-align)
  - Dead code elimination
  - Almost flat learning curve because it's very similar to classic CSS, but better
  - No (minimal) inline styling
  - Clean conditional statements

Styled components

- yarn add styled-components
- yarn add @types/styled-components
- Sofascore's preferred CSS in JS library
- Essentially writing styles inside template strings
    - Allows using JS variables inside those strings
- Can be in the same file as other JS or in separate files
- Docs

Styled components example - styled.md

Sofascore

02

SWR

Motivation

- Fetch + useEffect + useState is okayish, but we usually require complex features
    - E.g. polling, local caching, fetching on tab focus, request deduplication
    - We could implement this, but let's not reinvent the wheel 😉
- Solution: SWR (stale-while-revalidate)
    - **Hooks based** HTTP client library

SWR

- yarn add swr
    - Types are added out of the box
- Wrap your app in SWRConfig which accepts value of SWRConfiguration type
    - Fetcher callback needs to be defined
        - **For all intents and purposes of this Academy just c/p my demo**
- Check out App.tsx in our demo project
- e.g. const {data: match, error} = useSWR<MatchDetailsResponse>(matchRoute(matchId), {refreshInterval: 10000})
    - Store result of MatchDetailsResponse type into match variable, error in error variable
    - Poll the server every 10 seconds

SWR example - swr.md

Sofascore

03

Context

Motivation

- We have a value at the top of the app, and need it:
  - A) Pretty much everywhere (e.g. app's theme - light or dark)
  - B) In some component deep in the file tree
- We want to remain flexible where to use some data or logic
- Anti-solution: "prop drilling"
  - Pass everything via props to every component
  - The whole component tree re-renders on change, app is data heavy
- Solution: Context

Context

- Shipped with React
- Wrap your components/App with a ContextProvider component with some value (can be an object)
- Consume your Context via ContextConsumer component or useContext hook
  - Provider has to be rendered above consumer
  - Consumer will receive data from the nearest Provider of the same type (if multiple rendered)
- When Provider value changes, all its Consumers will rerender, to get new value.

Context vs Redux

- Redux is a 3rd party library for state-management
  - Extremely popular with React a few years ago, now it's just popular
  - Propagation of state changes to components
- Context comes out of the box with React
- Context is simpler to use and makes cleaner code in small apps
- Redux is more robust for state management and persisted states
  - Context can but isn't exactly intended to fully replace a state management library
- We at Sofascore use:
  - Context for passing data and avoiding prop drilling
  - Redux for global state management (e.g. reason: redux-saga)

Context example - context.md

Sofascore

Sofascore
Text FAN ASSIST to 95323 for assistance
Heineken

04

Routing

Routing

- URL addresses specific resource on the Internet (page, response, ...)
- Makes user navigation easier (refresh, browser back)
- Process of navigating to the specific resource on the web

Routing

SEO (Search Engine Optimization)

- Process of making a website more visible in search results
- Search engines crawl web, index content from pages, points to relevant pages in search results
- Ranking algorithm is secret, known only by search engine companies.

SEO (Search Engine Optimization)

- More traffic -> More 💰

Sofascore

Server side routing

- Route transition is handled on the server
  - When the URL changes, a new HTML document is retrieved from the server
- Browser has to communicate with the server
- Good:
  - Minimal data for each page
  - Search engine friendly
- Bad:
  - Slower interaction between pages
  - Full refresh of a page -> Context is lost
- Does not necessarily imply server-side rendering

Client side routing

- Route transition is handled on the client
    - When URL changes, new HTML is not needed, current one is changed
- Good:
    - Faster page transitions
    - Preserves context
    - Page transitions can be animated
- Bad:
    - Larger first load
    - Possible extra data
    - Not so friendly to search engines - they have to render the page like a true browser
- First load can be server-side rendered, but it isn't default in React

Thank you for your attention!

Sofascore