Sofascore Frontend Academy Lecture 07,
April, 2024

SWR. Linters. Refs. CSS in JS.

Alen Murtić

Sofascore

TABLE OF CONTENTS

01 SWR

02 Linters

03 Refs

04 CSS in JS

Sofascore

01

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

02

Linters

Motivation

- Javascript is dynamically typed and interpreted language with many quirks
- Some of JS quirks are solved by using Typescript, but it is not end-all, be-all
  - TS just describes types, but you can make many other mistakes in JS
- Solution: Linters
  - Wikipedia: “Lint is the computer science term for a static code analysis tool used to flag programming errors, bugs, stylistic errors and suspicious constructs… A program which performs this function is also known as a "linter”
  - Most of linting in e.g. Java or C# is done by the compiler, but JS is interpreted, not compiled

Linters

ESLint

- ~~TSLint - Typescript-specific linter - deprecated~~
- THE linter for Javascript and Typescript
- Extendable and customizable
    - You can do practically anything you want if you have enough time and willpower
- Not just for static code analysis, but also for auto code formatting and style
- Many, many, many predefined configurations
    - In a larger project, you will most likely need your own custom config
    - https://eslint.org/docs/latest/use/configure/
    - Even at Sofascore, every project has its own configuration because of different structures and technologies

03

Refs

Refs

- Two distinct usages:
  - Storing data which should not trigger re-renders
  - Manipulating DOM - but, more on that later
- Official docs
- Returns object with current key set to DOM object or null
- useRef hook - const ref = useRef(null)
- createRef method - this.ref = createRef(null)

Refs example - refs.md

Sofascore

usePrevious

- Sometimes we need previous prop value in current component
    - In class components componentDidUpdate could access it
    - Use case example: animations
- We don’t want to re-render on update of previous value
- Solution: usePrevious hook

Refs example - usePrevious.md

Sofascore

04

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 - reminds you of Tailwind CSS
  - CSS is focused on defining document-level stylesheets, not component-level
- 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 and Next 13 problems

- Styled components was Sofascore's favorite CSS in JS library - not just ours
  - Excellent development experience, with very minor performance downsides
- Essentially writing CSS inside template strings
  - Allows usage of JS variables in CSS
- The lib would analyze the whole page and create custom classes for best performance
  - Key phrase: whole page - analyzed on client, during runtime
- Next 13 and app router introduces server components
  - Styled components authors basically said "we ain't gonna support it" since it doesn't fit in the lib's philosophy
- Solution: another library with compile-time CSS

Next 13 and RSC compatible libraries

- Other libraries have added full support for RSC
  - vanilla-extract : https://vanilla-extract.style/
  - Kuma UI: https://www.kuma-ui.com/
  - Panda CSS: https://panda-css.com/
  - StyleX - https://stylexjs.com/
- We will be using Kuma UI in the next lesson

Styled components example - styled.md

Sofascore

Thank you for your attention!

Sofascore