Fetch, useEffect, React Query, SWR, what else?

Kontent.ai

Ondrej Polesny

Developer Evangelist
Kontent.ai

Kontent.ai

Agenda

- Actual data fetching
- Advanced features in data fetching
    - Error handling
    - Intercepting requests
    - Retry policy
- Processing the data on the app side
- Advanced use cases
    - Revalidation
    - Paging

Kontent.ai

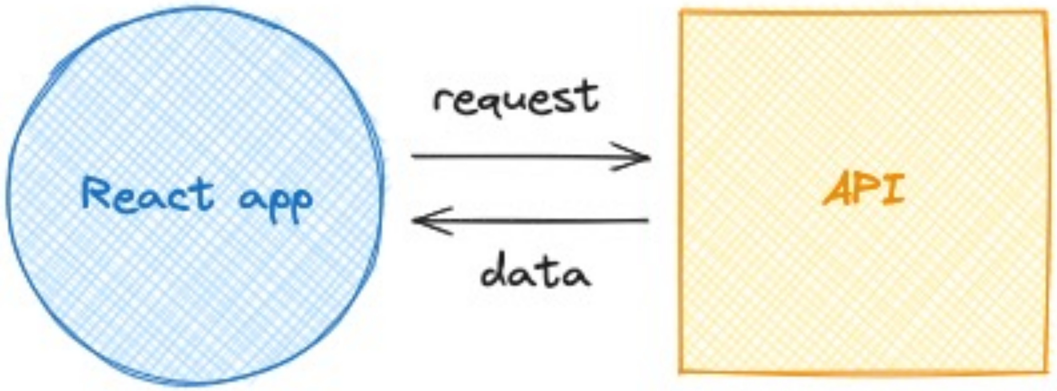
Frontend application showcase.

https://github.com/ondrabus/workshop-react-advanced-2023

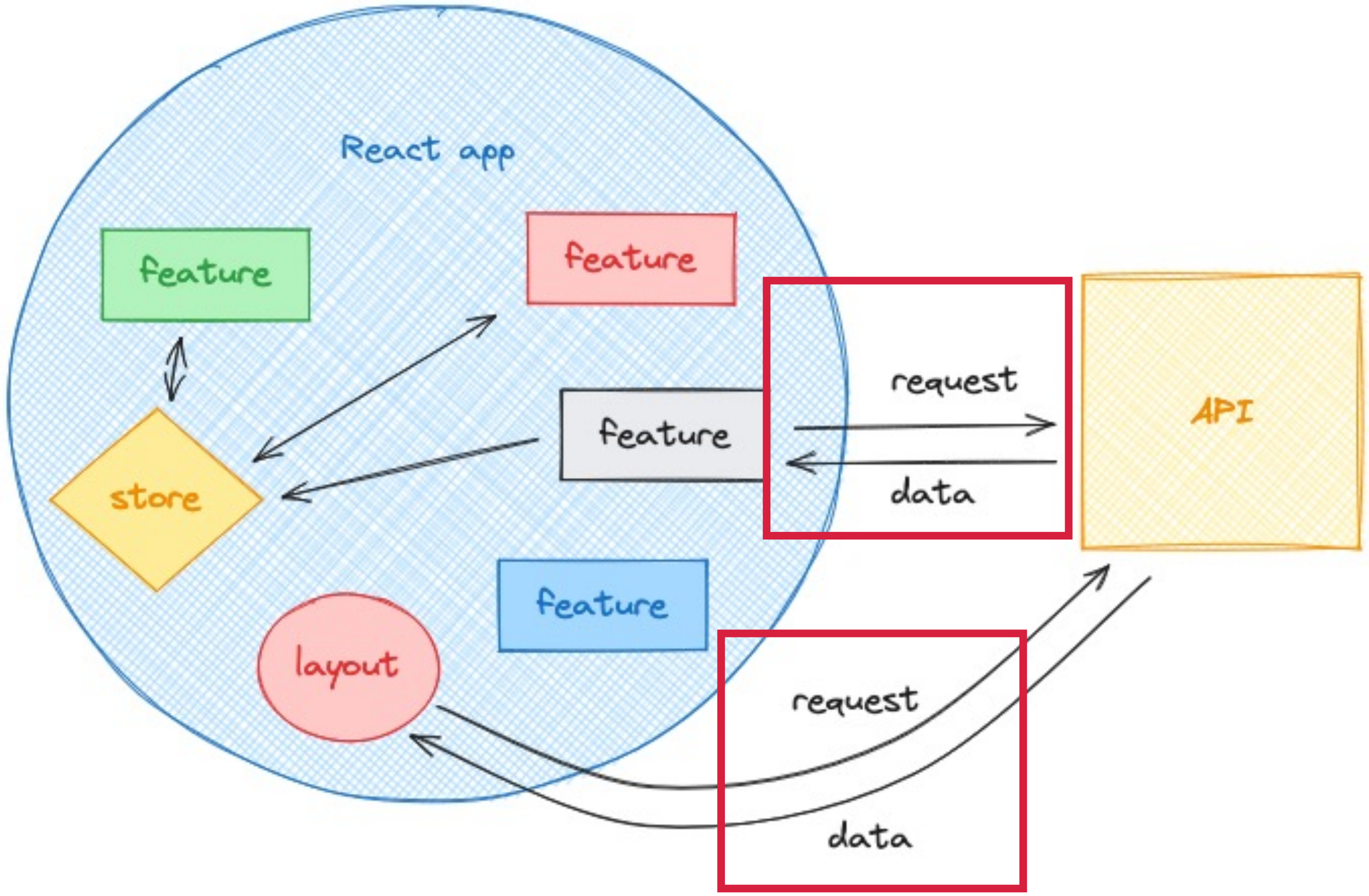
Kontent.ai

https://radv-workshop.netlify.app/api

Kontent.ai



Kontent.ai


React app
feature
feature
store
feature
feature
layout
request
data
API
request
data

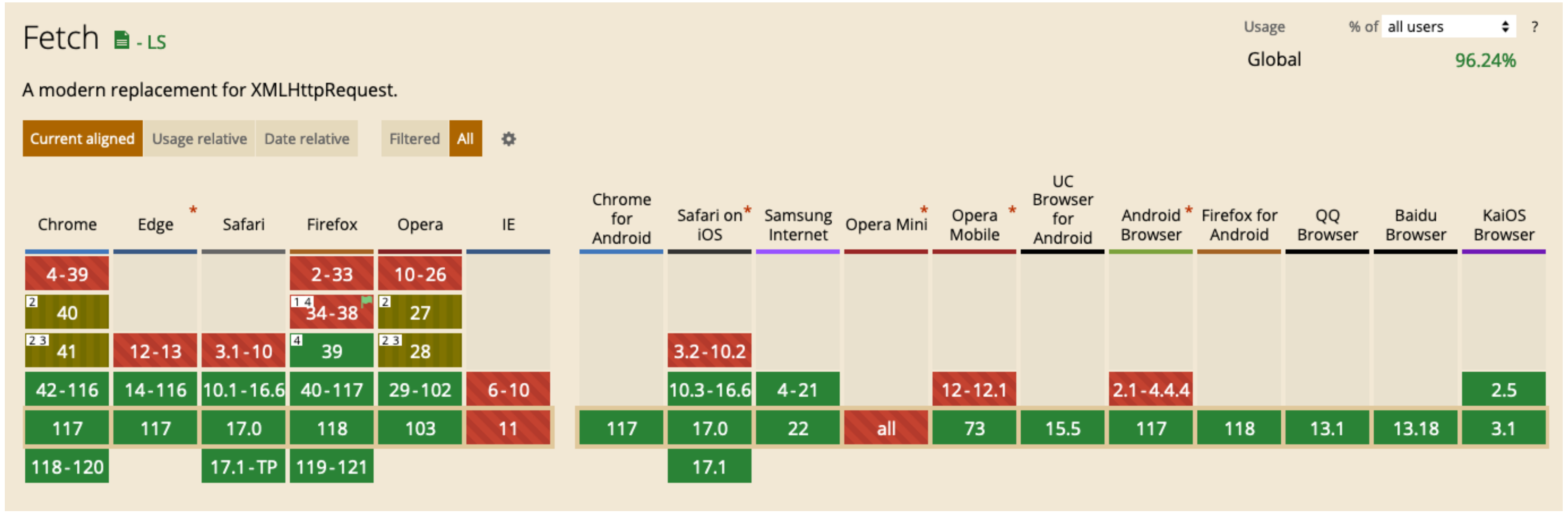
Kontent.ai

Fetching data

- XMLHttpRequest
- Axios
- Fetch API

Fetch API

- await fetch(”endpoint URL");
- Not available in old browsers (polyfill)
- Until early 2022 was not available in Node.js (Node.js <17.5)
- Native (no need for extra plugins)



Fetch - LS

Usage % of all users ?

Global 96.24%

A modern replacement for XMLHttpRequest.

Current aligned | Usage relative | Date relative | Filtered | All

Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-39			2-33	10-26												
[2] 40			[1 4] 34-38	[2] 27												
[2 3] 41	12-13	3.1-10	[4] 39	[2 3] 28			3.2-10.2									
42-116	14-116	10.1-16.6	40-117	29-102	6-10		10.3-16.6	4-21		12-12.1		2.1-4.4.4				2.5
117	117	17.0	118	103	11	117	17.0	22	all	73	15.5	117	118	13.1	13.18	3.1
118-120		17.1-TP	119-121				17.1									

Kontent.ai

Task #1

- Add the following fetch call to the components/Fetch.tsx

- `await (await fetch("/api/getFlights")).json()`

Kontent.ai

Axios

- Request and response interception
- Response timeout
- Retry policy
- Support for older browsers OOTB (based on XMLHttpRequest)
- Automatic JSON data transformation

Kontent.ai

Task #2

- Add the following Axios call to the components/Axios.tsx
- `await axios.get("/api/getFlights")`

Kontent.ai

Task #3 – intercepting request

- Add the following interceptor to the components/AxiosIntercept.tsx

```
axios.interceptors.request.use(config => {
    console.log(`Request was sent via Axios to
${config.url}.`);
    return config;
}, error => {
    return Promise.reject(error);
})
```

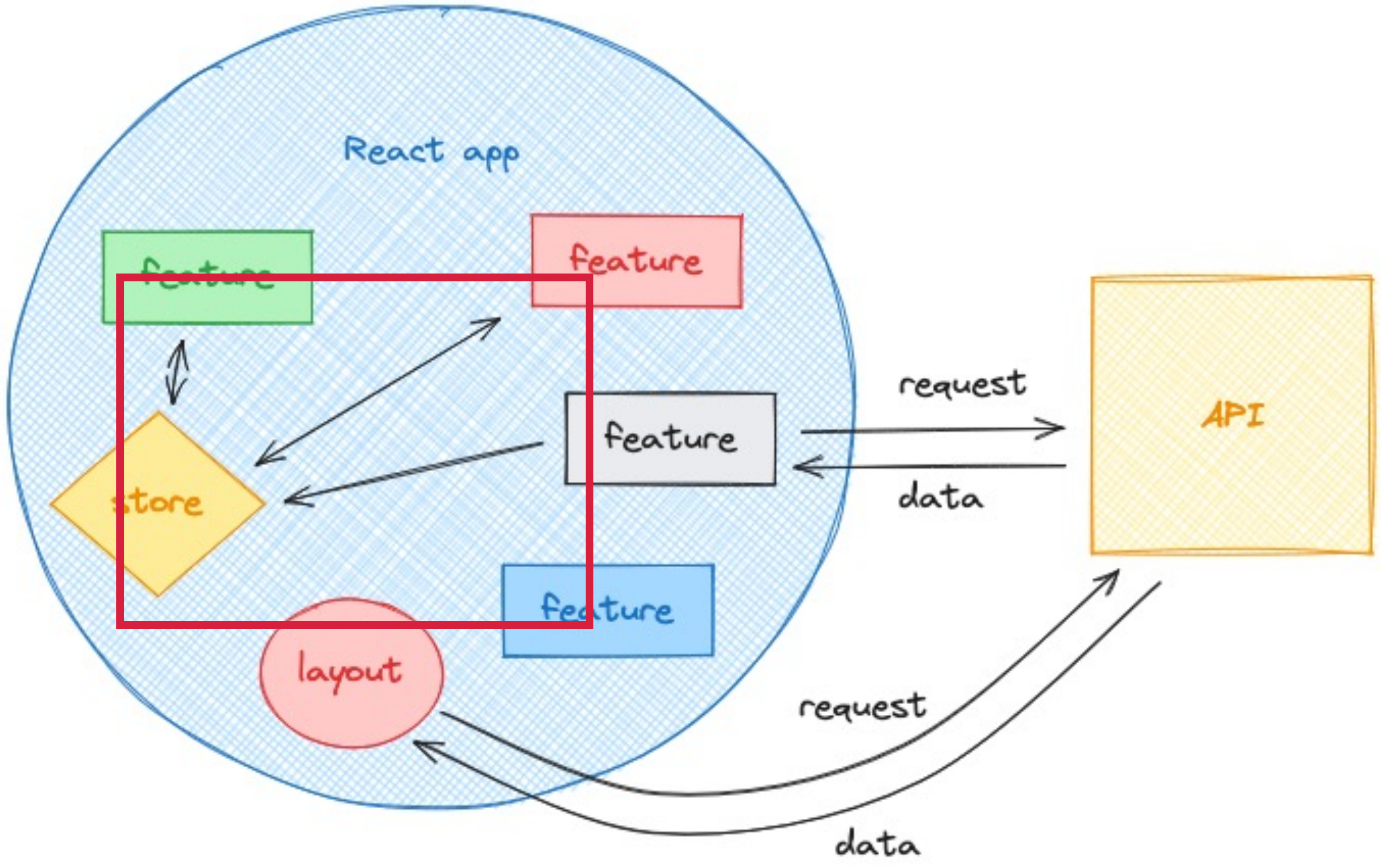
Kontent.ai

Task #4 – retry policy

- Add the following axios-retry configuration to the components/AxiosRetry.tsx

```
axiosRetry(axios, { retries: 10, retryDelay:
axiosRetry.exponentialDelay });
```

- Try to use the components/AxiosTimeout.tsx component to see the request time out


React app
feature
feature
feature
store
feature
layout
request
data
API
request
data

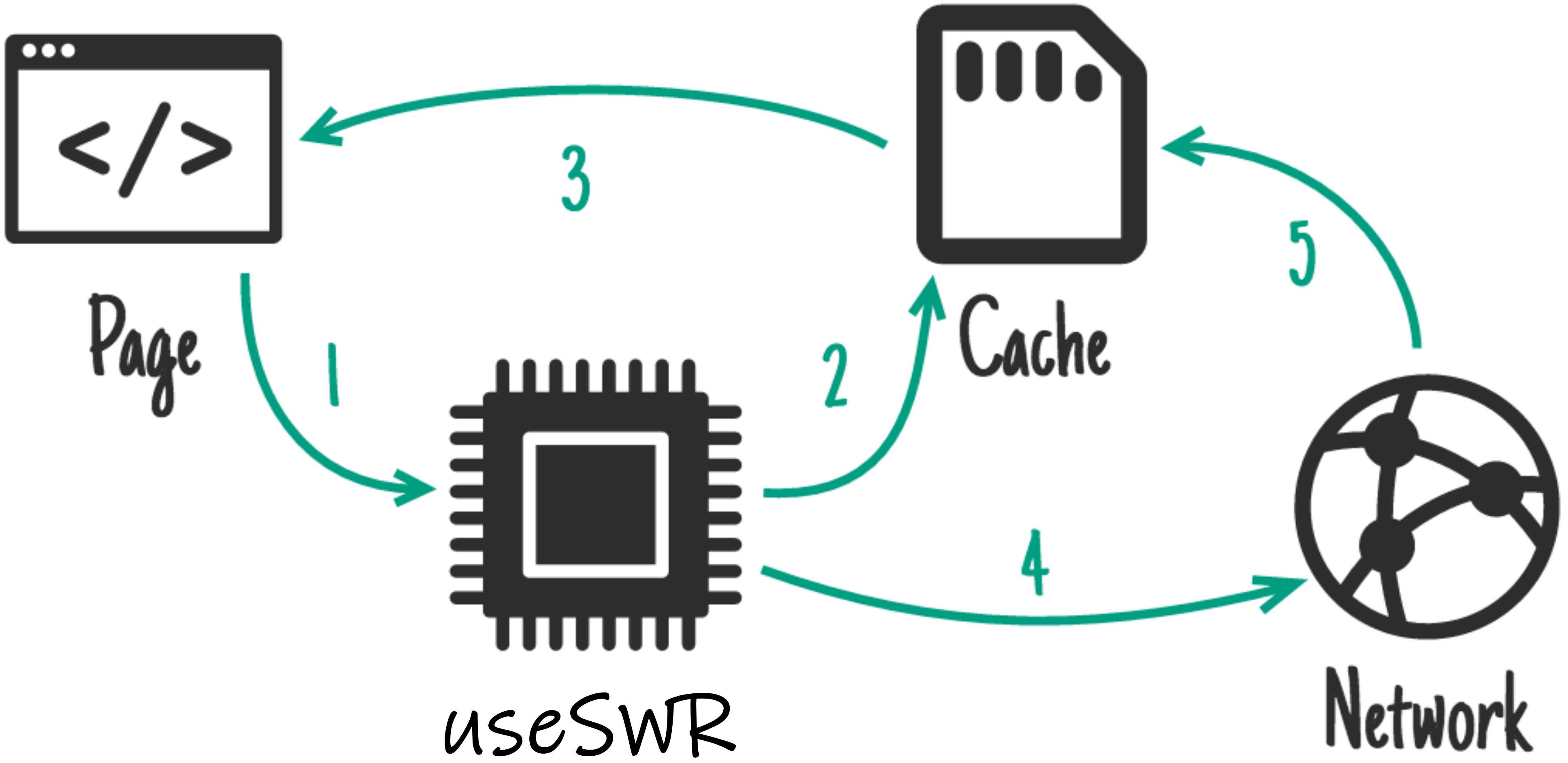
Kontent.ai

Fetching and managing the data

- useEffect
- useSWR
- useQuery

Kontent.ai

Stale While Revalidate



Kontent.ai

Image credit: https://developer.chrome.com/docs/workbox/modules/workbox-strategies/

SWR features

- Easy handling of loading/error states
- Easy configuration of retry policy
- Request deduplication
- Automatic/Interval-based/Interaction-based/No revalidation
- Ability to do paging and prefetching
- Bundle size 4.4kB

Kontent.ai

Task #5 – use SWR

- Use the useSWR hook instead of useEffect in the components/SWR.tsx

```
const {data, error, isLoading} = useSWR("flights",
flightsFetcher)
```

- Add state labels to let the user know when data are being fetched (loading) or error occurred
- Duplicate the component in App.tsx and check that the network request happens only once

Kontent.ai

Task #6 – mutate SWR

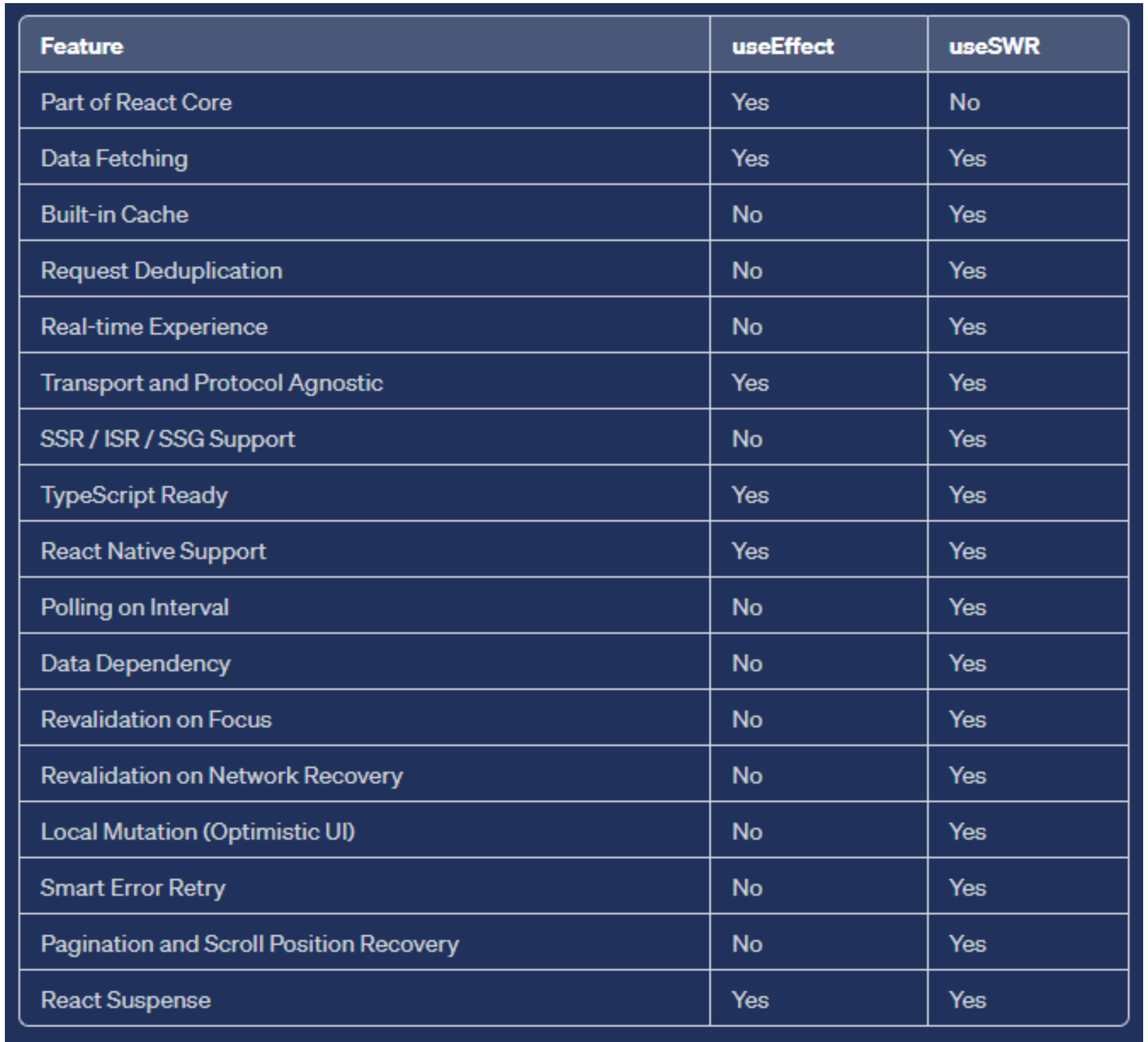
- Use the mutate function of SWR to invalidate the cached data (use the button in components/SWRMutate.tsx and add it to the App.tsx)

```
mutate("flights")
```

- Bonus: try the retry policy implemented in SWR by using the components/SWRRetry.tsx component

SWR vs useEffect

Image credit: https://www.linkedin.com/pulse/useeffect-vs-useswr-battle-hooks-sadegh-nakhjavani/



Feature	useEffect	useSWR
Part of React Core	Yes	No
Data Fetching	Yes	Yes
Built-in Cache	No	Yes
Request Deduplication	No	Yes
Real-time Experience	No	Yes
Transport and Protocol Agnostic	Yes	Yes
SSR / ISR / SSG Support	No	Yes
TypeScript Ready	Yes	Yes
React Native Support	Yes	Yes
Polling on Interval	No	Yes
Data Dependency	No	Yes
Revalidation on Focus	No	Yes
Revalidation on Network Recovery	No	Yes
Local Mutation (Optimistic UI)	No	Yes
Smart Error Retry	No	Yes
Pagination and Scroll Position Recovery	No	Yes
React Suspense	Yes	Yes

Kontent.ai

React Query

- Formerly known as ReactQuery, now called TanStack Query
- Very similar to useSWR, but has more features
- Automatic retry of failed requests
- Query cancellation
- Initial query data (SSG)

- Bundle size 13kB

Task #7 – use React Query

- Try ReactQuery instead of SWR in components/ReactQuery.tsx

```
const { isLoading, isError, isRefetching, data } =
useQuery({ queryKey: ["flights"], queryFn:
flightsFetcher})
```

- Add state labels to let the user know when data are being fetched (loading), refetched, or error occurred

Kontent.ai

How to choose the best one?

Kontent.ai

Get in touch.

Ondrej Polesny

Developer Evangelist
Kontent.ai

https://twitter.com/ondrabus

Kontent.ai