

VUEJS AMSTERDAM 2024

EFFICIENT SERVER STATE MANAGEMENT

WHO AM I?

- My name is Elise Patrikainen
- Live in Paris
- Senior front-end freelance developer
- Working at Accor
- Co-organiser of the meetup Vue.js Paris



WHO AM I?

- My name is Elise Patrikainen
 - Live in Paris
 - Senior front-end freelance developer
 - Working at Accor
 - Co-organiser of the meetup Vue.js Paris
- => looking for speakers



WHAT IS SERVER STATE ?

Spoiler: this is not about SSR.

WHAT IS STATE ?

State is made of:

- data fetch from the server
- data owned by our application

WHAT IS STATE ?

State is made of:

- data fetch from the server => **server state**
- data owned by our application => **client state**

SERVER STATE MANAGEMENT

EXAMPLE

Home Product Features Marketplace Company A propos Basket X

Dashboard

- > Teams
- > Projects
- Calendar
- Documents
- Reports

Basic X (Black) 24 € Basic Tee (Sienna) 24 € Basic G (Grey) 24 € Basic Pink (Pink) 288 €

Basic X
Black
1

7 left

Basic Tee
Sienna
0

3 left

Basic G
Grey
0

11 left

Basic Pink
Pink
1

3

Order summary

Basic X (Black) x 1

Basic Pink (Pink) x 1

Outside (Black) x 1

Basic (White) x 1

Order total 288 €

Checkout

Sea (White) 150 € Outside (Black) 210 € Promenade (Beige) 200 € Satchel (Blue) 2

Sea
White
0

6 left

Outside
Black
1

8 left

Promenade
Beige
0

9 left

Satchel
Blue
0

4

GLOBAL STATE = SERVER STATE + CLIENT STATE

Fetch from
the server

Belongs to
the client

SERVER STATE MANAGEMENT

SERVER STATE

CLIENT STATE

Asynchronous

=> fetch (loading / error)

=> cache

Synchronous

Shared ownership

=> refresh

Single ownership

- ▶ A concrete example of how quickly server state can become complex :



SERVER STATE MANAGEMENT

- ▶ A concrete example of how quickly server state can become complex :

Natalia Tepluhina - Local state and server cache: how to balance them with vue-query

```
1 actions: {
2   async fetchTractor(id) {
3     this.loading = true
4     try {
5       const response = await getTractor(id)
6       const existingTractor = this.tractors.find((t) => t.id === id)
7       if (existingTractor) {
8         const { description, engine, rated_power } = response.data
9         existingTractor.description = description
10        existingTractor.engine = engine
11        existingTractor.rated_power = rated_power
12      } else {
13        this.tractors.push(response.data)
14      }
15    } catch (error) {
16      this.error = error
17    }
18    this.loading = false
19  },
20},
```

COPY

Quitter le mode plein écran (f)

Faites défiler la page pour afficher plus de détails

6:22 / 27:58

WHY IS SERVER STATE MANAGEMENT SO HARD ?

- Handling states related to the fetch (error, loading)?
- Request deduplications?
- How to cache the data?
- When to refresh the data? On component mount? Polling? On window focus? On reconnection?
- Garbage collecting the inactive cache?

2 PATTERNS:

- ▶ server state in component scope
=> no cache

- ▶ server state in a store
=> no refresh

SERVER STATE MANAGEMENT

So what can we do ?

SERVER STATE MANAGEMENT

We can use a server state management library 

SERVER STATE MANAGEMENT

- ▶ One of them belongs to this place:



- ▶ One of them belongs to this place:



TANSTACK



- ▶ A set of open source libraries
(by Tanner Linsley)
- ▶ Framework agnostic



TanStack

High-quality open-source software for web developers.

Headless, type-safe, & powerful utilities for State Management, Routing, Data Visualization, Charts, Tables, and more.

Open Source Libraries

TanStack Query
Powerful asynchronous state management, server-state utilities and data fetching
Fetch, cache, update, and wrangle all

TanStack Table
Headless UI for building powerful tables & datagrids
Supercharge your tables or build a datagrid from scratch for TS/JS, React,

TanStack Router
Type-safe Routing for React applications.
Powerful routing for your React applications including a fully type-safe



- ▶ A set of open source libraries
(by Tanner Linsley)
- ▶ Framework agnostic

TANSTACK QUERY



TanStack

High-quality open-source software for web developers.

Headless, type-safe, & powerful utilities for State Management, Routing, Data Visualization, Charts, Tables, and more.

Open Source Libraries

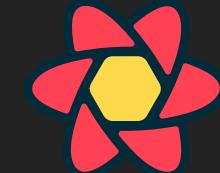
TanStack Query
Powerful asynchronous state management, server-state utilities and data fetching
Fetch, cache, update, and wrangle all

TanStack Table
Headless UI for building powerful tables & datagrids
Supercharge your tables or build a datagrid from scratch for TS/JS, React,

TanStack Router
Type-safe Routing for React applications.
Powerful routing for your React applications including a fully type-safe



SHORT HISTORY

- ▶ ReactQuery 
- => Vue Query (wrapper) 
- ▶ Migrated to TanStack Query
 - JS core
 - frameworks adaptations



TanStack Query v5

Powerful asynchronous state management for TS/JS, React, Solid, Vue, Svelte and Angular

Toss out that granular state management, manual refetching and endless bowls of async-spaghetti code. TanStack Query gives you declarative, always-up-to-date auto-managed queries and mutations that **directly improve both your developer and user experiences**.

[READ THE DOCS](#)

(or check out [query.gg](#) – the official React Query course)

TABLE OF CONTENTS:

- ▶ Overview of Tanstack Query
- ▶ Demo of Tanstack Query
- ▶ Alternatives to Tanstack Query

OVERVIEW OF TANSTACK QUERY



3 NOTIONS:

- ▶ Queries: fetch the data 
- ▶ Mutations : mutate the data 
- ▶ QueryClient: queries and mutations state 



TANSTACK QUERY – OVERVIEW

► Queries:



- ▶ **Queries:**

- ▶ Declaring a query:



- ▶ **Queries:**

- ▶ Declaring a query:

```
const { isPending, isError, data } = useQuery({  
  queryKey: ['items'],  
  queryFn: () => fetch<Item[]>(`https://.../products`),  
  staleTime: 60 * 1000  
})
```



- ▶ **Queries:**

- ▶ Declaring a query:

```
const { isPending, isError, data } = useQuery({  
  queryKey: ['items'],  
  queryFn: () => ofetch<Item[]>(`https://.../products`),  
  staleTime: 60 * 1000  
})
```



- ▶ **Queries:**

- ▶ Declaring a query:

```
const { isPending, isError, data } = useQuery({  
  queryKey: ['items'],  
  queryFn: () => ofetch<Item[]>(`https://.../products`),  
  staleTime: 60 * 1000  
})
```



TANSTACK QUERY – OVERVIEW

- ▶ **Queries:**
 - ▶ Declaring a query:



- ▶ **Queries:**

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):



► Queries:

- Declaring a query:
- OptionQuery API (v5):

```
const { isPending, isError, data } = useQuery({  
  queryKey: ['items'],  
  queryFn: () => ofetch<Item[]>(`https://.../products`),  
  staleTime: 60 * 1000  
})
```



► Queries:

- Declaring a query:
- OptionQuery API (v5):

```
const itemsQuery = queryOptions({
```

```
})
```

```
const { isPending, isError, data } = useQuery({  
  queryKey: ['items'],  
  queryFn: () => ofetch<Item[]>(`https://.../products`),  
  staleTime: 60 * 1000  
})
```



► Queries:

- Declaring a query:
- OptionQuery API (v5):

```
const itemsQuery = queryOptions({  
  queryKey: ['items'],  
  queryFn: () => ofetch<Item[]>(`https://.../products`),  
  staleTime: 60 * 1000  
})
```

```
const { isPending, isError, data } = useQuery({  
})
```



► Queries:

- Declaring a query:
- OptionQuery API (v5):

```
const itemsQuery = queryOptions({  
  queryKey: ['items'],  
  queryFn: () => fetch<Item[]>(`https://.../products`),  
  staleTime: 60 * 1000  
})
```

```
const { isPending, isError, data } = useQuery(itemsQuery)
```



► Queries:

- Declaring a query:
- OptionQuery API (v5):

```
const itemsQuery = queryOptions({  
  queryKey: ['items'],  
  queryFn: () => fetch<Item[]>(`https://.../products`),  
  staleTime: 60 * 1000  
})
```

```
const { isPending, isError, data } = useQuery(itemsQuery)
```



▶ Queries:

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

```
const itemsQuery = queryOptions({  
  queryKey: ['items'],  
  queryFn: () => fetch<Item[]>(`https://.../products`),  
  staleTime: 60 * 1000  
})
```

```
const { isPending, isError, data } = useQuery(itemsQuery)
```



- ▶ **Queries:**

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):



- ▶ **Queries:**

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

- Increase queries reusability 



- ▶ **Queries:**

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

- Increase queries reusability 
- Improves type safety 



▶ Queries:

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):



- ▶ **Queries:**

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

- ▶ **Mutations:**



- ▶ **Queries:**

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

- ▶ **Mutations:**

```
const { isPending, isSuccess, mutate } = useMutation({  
  mutationFn: (body: any) => fetch('/comment', { method: "POST", body }),  
  onError() {  
    displayErrorSnackbar()  
  }  
})
```



▶ Queries:

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

▶ Mutations:

```
const { isPending, isSuccess, mutate } = useMutation({  
  mutationFn: (body: any) => fetch('/comment', { method: "POST", body }),  
  onError() {  
    displayErrorSnackbar()  
  }  
})
```



- ▶ **Queries:**

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

- ▶ **Mutations:**

```
const { isPending, isSuccess, mutate } = useMutation({  
  mutationFn: (body: any) => fetch('/comment', { method: "POST", body }),  
  onError() {  
    displayErrorSnackbar()  
  }  
})
```



► Queries:

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

► Mutations:

```
const { isPending, isSuccess, mutate } = useMutation({  
  mutationFn: (body: any) => fetch('/comment', { method: "POST", body }),  
  onError() {  
    displayErrorSnackbar()  
  }  
})
```



DECLARATIVE



IMPERATIVE



▶ Queries:

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

▶ Mutations:

```
const { isPending, isSuccess, mutate } = useMutation({  
  mutationFn: (body: any) => fetch('/comment', { method: "POST", body }),  
  onError() {  
    displayErrorSnackbar()  
  }  
})
```



▶ Queries:

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

▶ Mutations:

```
const { isPending, isSuccess, mutate } = useMutation({  
  mutationFn: (body: any) => fetch('/comment', { method: "POST", body }),  
  onError() {  
    displayErrorSnackbar()  
  }  
})  
  
function addComment(comment: string) {  
  mutate({ content: comment })  
}
```



▶ Queries:

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

▶ Mutations:

```
const { isPending, isSuccess, mutate } = useMutation({  
  mutationFn: (body: any) => fetch('/comment', { method: "POST", body }),  
  onError() {  
    displayErrorSnackbar()  
  }  
})  
  
function addComment(comment: string) {  
  mutate({ content: comment })  
}
```



- ▶ **Queries:**

- ▶ Declaring a query:
- ▶ OptionQuery API (v5):

- ▶ **Mutations:**



- ▶ **Queries:**
 - ▶ Declaring a query:
 - ▶ OptionQuery API (v5):
- ▶ **Mutations:**
- ▶ **QueryClient:**



- ▶ **Queries:**
 - ▶ Declaring a query:
 - ▶ OptionQuery API (v5):
- ▶ **Mutations:**
- ▶ **QueryClient:**

```
const queryClient = useQueryClient()
```



- ▶ **Queries:**
 - ▶ Declaring a query:
 - ▶ OptionQuery API (v5):
- ▶ **Mutations:**
- ▶ **QueryClient:**

```
const queryClient = useQueryClient()  
queryClient.prefetchQuery(itemsQuery)  
queryClient.setQueryData(itemsQuery.queryKey, [...oldItems, newItem])  
queryClient.invalidateQueries(itemsQuery)
```



- ▶ **Queries:**
 - ▶ Declaring a query:
 - ▶ OptionQuery API (v5):
- ▶ **Mutations:**
- ▶ **QueryClient:**

```
const queryClient = useQueryClient()
queryClient.prefetchQuery(itemsQuery)
queryClient.setQueryData(itemsQuery.queryKey, [...oldItems, newItem])
queryClient.invalidateQueries(itemsQuery)
```



- ▶ **Queries:**
 - ▶ Declaring a query:
 - ▶ OptionQuery API (v5):
- ▶ **Mutations:**
- ▶ **QueryClient:**

```
const queryClient = useQueryClient()
queryClient.prefetchQuery(itemsQuery)
queryClient.setQueryData(itemsQuery.queryKey, [...oldItems, newItem])
queryClient.invalidateQueries(itemsQuery)
```



- ▶ **Queries:**
 - ▶ Declaring a query:
 - ▶ OptionQuery API (v5):
- ▶ **Mutations:**
- ▶ **QueryClient:**

```
const queryClient = useQueryClient()
queryClient.prefetchQuery(itemsQuery)
queryClient.setQueryData(itemsQuery.queryKey, [...oldItems, newItem])
queryClient.invalidateQueries(itemsQuery)
```



- ▶ **Queries:**
 - ▶ Declaring a query:
 - ▶ OptionQuery API (v5):
- ▶ **Mutations:**
- ▶ **QueryClient:**

```
const queryClient = useQueryClient()
queryClient.prefetchQuery(itemsQuery)
queryClient.setQueryData(itemsQuery.queryKey, [...oldItems, newItem])
queryClient.invalidateQueries(itemsQuery)
```

DEMO

TANSTACK QUERY ALTERNATIVES

TANSTACK QUERY ALTERNATIVES

VUE APOLLO



TANSTACK QUERY ALTERNATIVES

VUE APOLLO



PINIA COLADA (WIP)

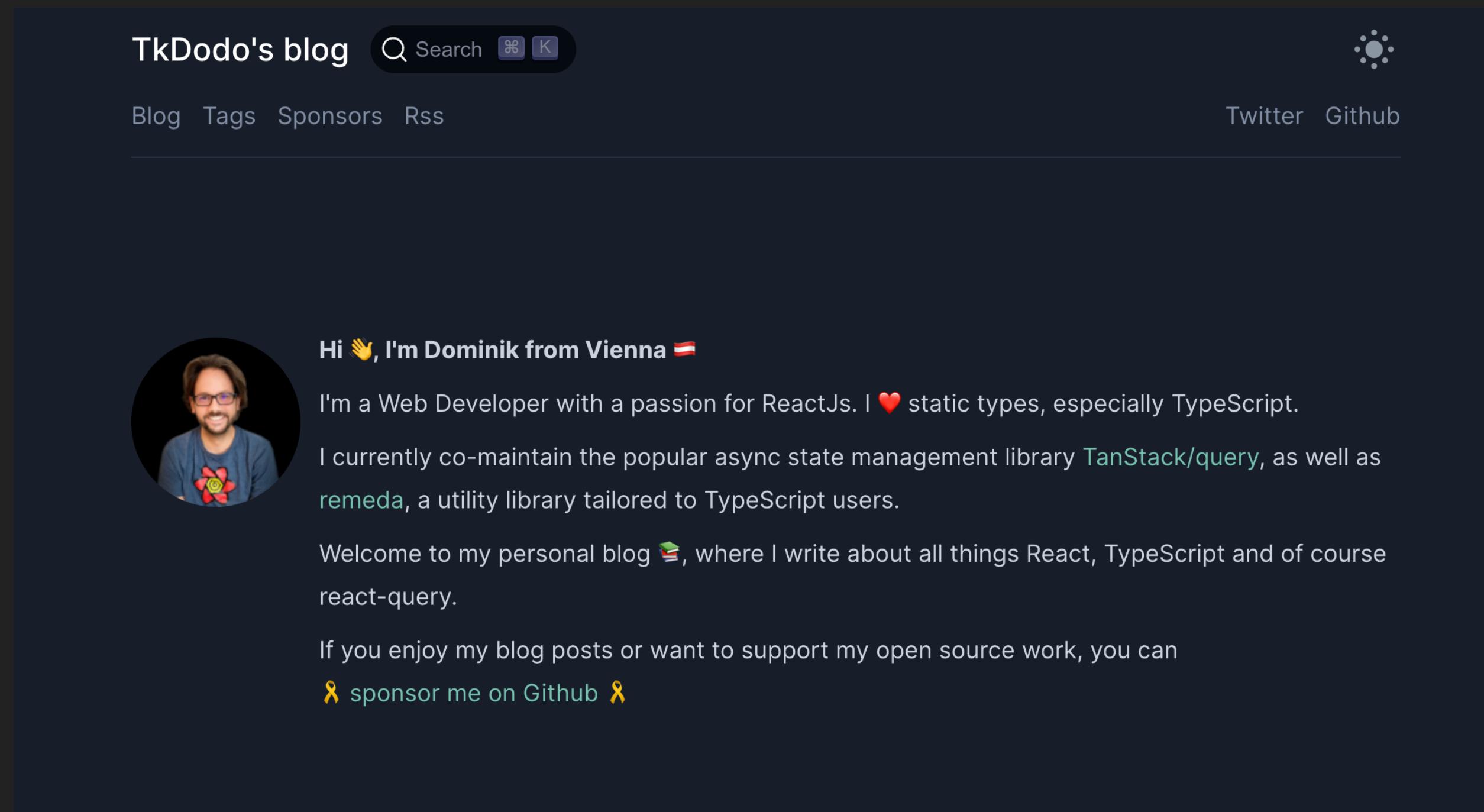


RESOURCES

RESOURCES

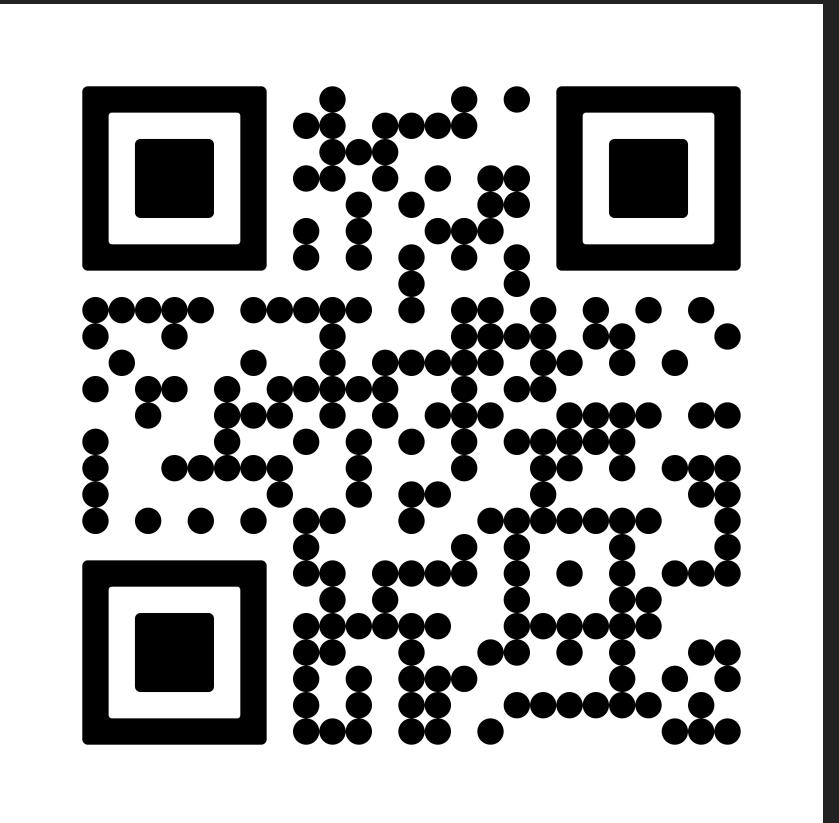
TK DODO'S BLOG

► <https://tkdodo.eu/blog/>

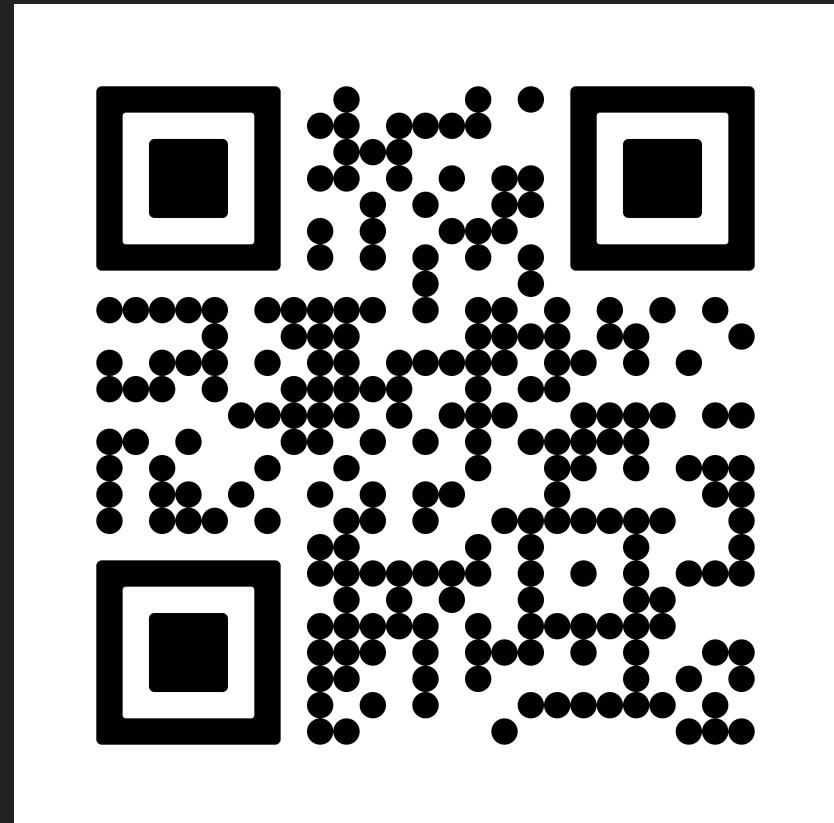
A screenshot of the TkDodo's blog homepage. The header features the title "TkDodo's blog" and a search bar with icons for magnifying glass, file, and keyboard. Below the header are navigation links for "Blog", "Tags", "Sponsors", and "Rss". On the right side of the header are links for "Twitter" and "Github". A circular profile picture of a man with glasses and a beard is on the left. The main content area starts with a greeting: "Hi 🙌, I'm Dominik from Vienna 🇦🇹". It then describes the author as a Web Developer with a passion for ReactJs, static types, especially TypeScript. It mentions co-maintaining the popular async state management library [TanStack/query](#), as well as the utility library [remeda](#), tailored to TypeScript users. The text continues with a welcome message about the blog, its focus on React, TypeScript, and react-query, and an invitation to sponsor the author on GitHub.

RESOURCES

DEMO



SLIDES



- ▶ Demo: <https://github.com/ElisePatrikainen/demo-vuejs-amsterdam-2024>
- ▶ Slides: <https://github.com/ElisePatrikainen/slides-vuejs-amsterdam-2024>

THANK YOU