how to get started with tanstack store to store a user document from firestore

Below is a **practical, minimal way to get started with TanStack Store** and use it to hold a **user document from Firestore**. I'll assume:

- You're using **React**
- You already have **Firebase / Firestore set up**
- You want a **global, reactive user store**

---

1. Install TanStack Store

bash

```
npm install @tanstack/store @tanstack/react-store
```

Copy code

- `@tanstack/store` → core store
- `@tanstack/react-store` → React bindings

---

2. Create a User Store

Create a file like `stores/userStore.ts`.

ts

```ts
import { Store } from '@tanstack/store'

export type UserDoc = {
  uid: string
  email: string
  displayName?: string
  role?: string
}
```

```ts
type UserState = {
  user: UserDoc | null
  loading: boolean
}

export const userStore = new Store<UserState>({
  user: null,
  loading: true,
})
```

Copy code

This store:

- Holds the Firestore user document
- Tracks loading state

---

3. Create Actions to Update the Store

TanStack Store is **action-driven** (no reducers).

ts

```ts
export const setUser = (user: UserDoc | null) => {
  userStore.setState((state) => ({
    ...state,
    user,
    loading: false,
  }))
}

export const setLoading = (loading: boolean) => {
  userStore.setState((state) => ({
    ...state,
    loading,
  }))
}
```

Copy code

---

4. Subscribe to Firestore User Document

Create a Firestore listener (e.g. `lib/subscribeToUser.ts`):

ts

```ts
import { doc, onSnapshot } from 'firebase/firestore'
import { db } from './firebase'
import { setUser, setLoading } from '../stores/userStore'

export const subscribeToUser = (uid: string) => {
  setLoading(true)

  const ref = doc(db, 'users', uid)

  return onSnapshot(ref, (snapshot) => {
    if (!snapshot.exists()) {
      setUser(null)
      return
    }

    setUser({
      uid,
      ...(snapshot.data() as any),
    })
  })
}
```

Copy code

This keeps TanStack Store **in sync with Firestore in real time**.

---

5. Connect It to Firebase Auth

Example in `authListener.ts`:

ts

```ts
import { onAuthStateChanged } from 'firebase/auth'
import { auth } from './firebase'
import { setUser, setLoading } from '../stores/userStore'
import { subscribeToUser } from './subscribeToUser'

let unsubscribeUserDoc: (() => void) | null = null
```

```
export const initAuthListener = () => {
  setLoading(true)

  onAuthStateChanged(auth, (user) => {
    if (unsubscribeUserDoc) {
      unsubscribeUserDoc()
      unsubscribeUserDoc = null
    }

    if (!user) {
      setUser(null)
      return
    }

    unsubscribeUserDoc = subscribeToUser(user.uid)
  })
}
```

Copy code

Call this **once** when your app starts.

tsx

Copy code

```
import { useStore } from '@tanstack/react-store'
import { userStore } from '../stores/userStore'

export function UserProfile() {
  const { user, loading } = useStore(userStore)

  if (loading) return <p>Loading...</p>
  if (!user) return <p>Not signed in</p>

  return (
    <div>
      <p>Email: {user.email}</p>
      <p>Role: {user.role}</p>
    </div>
```

```
    )
  }
```

✅ Component re-renders automatically when Firestore updates.

---

7. App Initialization

In `main.tsx` or `App.tsx`:

ts Copy code

```ts
import { useEffect } from 'react'
import { initAuthListener } from './lib/authListener'

export function App() {
  useEffect(() => {
    initAuthListener()
  }, [])

  return <YourRoutes />
}
```

Why TanStack Store Works Well for Firestore

Advantages

- Simple, explicit updates
- No boilerplate reducers
- Perfect for **external subscriptions** (Firestore, WebSockets)
- Framework-agnostic core

Mental model

> Firestore is the source of truth → TanStack Store mirrors it → React reads from the store

---

Common Pattern Enhancements

- Add `hasHydrated` flag for SSR

- Split auth user vs profile document
- Add derived stores for permissions

If you want, tell me:

- **React / Next.js / Expo?**
- **Auth + user doc structure**
- **SSR or client-only**

and I'll tailor this to your exact setup.

where do I put these functions

```
export const setUser = (user: UserDoc | null) => {
  userStore.setState((state) => ({
    ...state,
    user,
    loading: false,
  }))
}

export const setLoading = (loading: boolean) => {
  userStore.setState((state) => ({
    ...state,
    loading,
  }))
}
```

You should put those functions **in the same file where you define your store** (or in a closely related file for store actions). This keeps your store and the functions that update it together, which is a common pattern in TanStack Store.

Here's a good folder/file structure example:

bash

Copy code

```
/stores
  └ userStore.ts
/lib
  └ firebase.ts       (firebase/firestore setup)
/lib
  └ subscribeToUser.ts
/lib
  └ authListener.ts
```

---

stores/userStore.ts

ts

Copy code

```ts
import { Store } from '@tanstack/store'

// 1. Define the user type
export type UserDoc = {
  uid: string
  email: string
  displayName?: string
  role?: string
}

// 2. Define the store state type
type UserState = {
  user: UserDoc | null
  loading: boolean
}

// 3. Create the store
export const userStore = new Store<UserState>({
  user: null,
  loading: true,
})

// 4. Actions to update the store
export const setUser = (user: UserDoc | null) => {
  userStore.setState((state) => ({
```

```
    ...state,
    user,
    loading: false,
  }))
}

export const setLoading = (loading: boolean) => {
  userStore.setState((state) => ({
    ...state,
    loading,
  }))
}
```

✅ **Why this works well:**

- `userStore` is the **central reactive state**
- `setUser` and `setLoading` are **helper functions (actions)** to update it safely
- Other files (like your Firestore subscription or auth listener) just **import and call these functions**

If you want, I can make a **full working example** showing **store + Firestore subscription + React component** all wired together, so you can just drop it in. It'll be like a copy-paste starter kit.

Do you want me to do that?