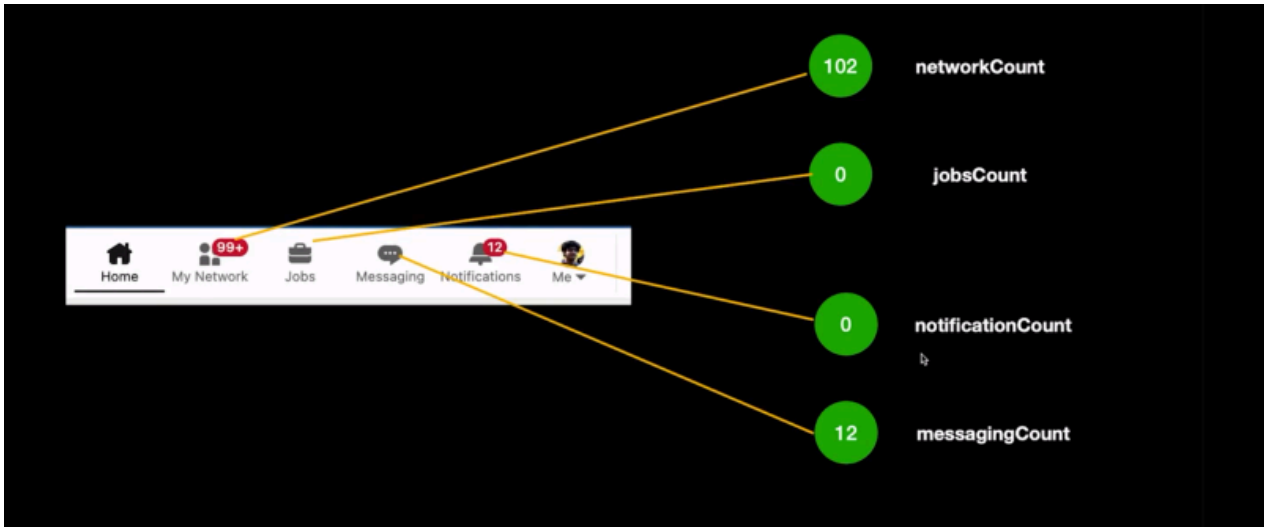
Recoil Deep Dive

Atoms, selectors, Asynchronous data queries useRecoilState, useRecoilValue, useSetRecoilState

atomFamily, selectorFamily, useRecoilStateLoadable, useRecoilValueLoadable

Atom



Dynamic part of it.
How to store these incase of react
We will use **useState()**

But if we want to do state management using **Recoil** then we will use **atom .**
An atom is smallest unit of state we can store similar to useState()
Here we have defined four atoms.

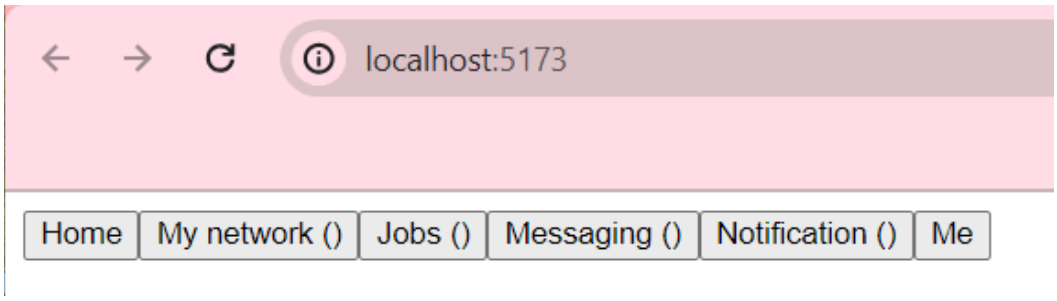
- networkCount
- jobsCount
- notificationCount
- messagingCount

npm vite create@latest

Now we will define these four atoms and make sort of LinkedIn topbar

App.jsx

```
function App() {
  return (
    <>
        <button>Home</button>

        <button>My network ()</button>
        <button>Jobs ()</button>
        <button>Messaging ()</button>
        <button>Notification ()</button>

        <button>Me</button>
    </>
  )
}

export default App
```



We will use atoms

Atom.js

```
import {atom} from "recoil"
export const networkAtom = atom({
```

```
    key: "networkAtom",
    default: 102
});
export const jobsAtom = atom({
    key: "jobsAtom",
    default:0
})
export const notificationsAtom = atom({
    key:"notificationsAtom",
    default:12
})
export const messagingAtom = atom({
    key:"messagingAtom",
    default:0
})
```

Now we will use one of the three primary recoil hooks

- useRecoilState
- useRecoilValue
- useSetRecoilState

Make sures the key you are defining to each atoms are unique else it will overwrite the previous ones.

To debug error We will use of console.log

```
import { useRecoilValue,RecoilRoot } from "recoil"
import { jobsAtom, messagingAtom, networkAtom, notificationsAtom } from
"./atom"
// RecoilRoot , Every component that uses Recoil need to be wrapped under
it

function App(){
  return <RecoilRoot>
```

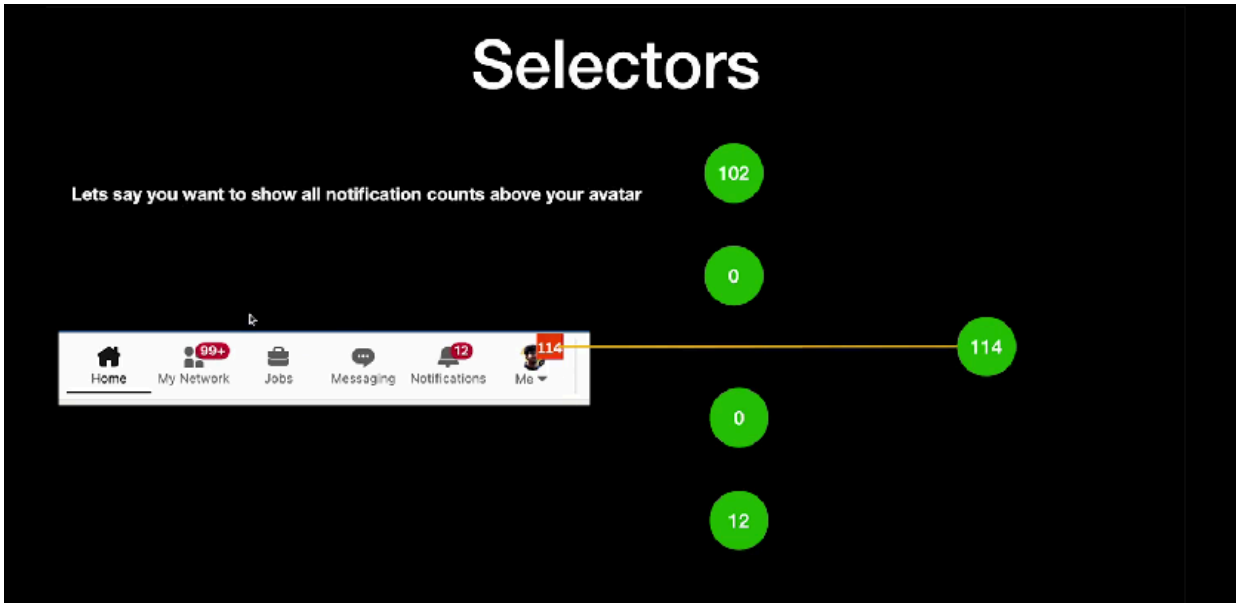
```
    <MainApp />
  </RecoilRoot>
}

function MainApp() {
  const networkNotificationCount = useRecoilValue(networkAtom)
  const jobsAtomCount = useRecoilValue(jobsAtom);
  const notificationsAtomCount = useRecoilValue(notificationsAtom);
  const messaginAtomCount = useRecoilValue(messagingAtom)

  return (
    <>
        <button>Home</button>

        <button>My network ({networkNotificationCount >=
100?"99+":networkNotificationCount})</button>
        <button>Jobs ({jobsAtomCount >= 100?"99+":jobsAtomCount})</button>
        <button>Messaging ({notificationsAtomCount >=
100?"99+":notificationsAtomCount})</button>
        <button>Notification ({messaginAtomCount >=
100?"99+":messaginAtomCount})</button>

        <button>Me</button>
    </>
  )
}

export default App
```

We added some logic into it

```
import { useRecoilValue,RecoilRoot,useSetRecoilState} from "recoil"
import { jobsAtom, messagingAtom, networkAtom, notificationsAtom } from
"./atom"
```

```
// RecoilRoot , Every component that uses Recoil need to be wrapped under
it

function App(){
  return <RecoilRoot>
    <MainApp />
  </RecoilRoot>
}

function MainApp() {
  const networkNotificationCount = useRecoilValue(networkAtom)
  const jobsAtomCount = useRecoilValue(jobsAtom);
  const notificationsAtomCount = useRecoilValue(notificationsAtom);
  const messagingAtomCount = useRecoilValue(messagingAtom);
  // gives two thing ,actual value and way to update that value.

  return (
    <>
        <button>Home</button>

        <button>My network ({networkNotificationCount >=
100?"99+":networkNotificationCount})</button>
        <button>Jobs ({jobsAtomCount >= 100?"99+":jobsAtomCount})</button>
        <button>Messaging ({messagingAtomCount >=
100?"99+":messagingAtomCount})</button>
        <button>Notification ({notificationsAtomCount >=
100?"99+":notificationsAtomCount})</button>

        {/* <button onClick={()=>{
          setMessagingAtomCount(messagingAtomCount + 1);
        }}>Me</button> */}

        {/* Clicking on this button will increase the messaging count . In
real world it will be done by something in backend by doing backend
call*/}
```

```
        {/* Now lets see a use case where we will use useSetRecoilState,
we will now make a separate component to update the notification count */}
        <ButtonUpdater/>
    </>
  )
}

function ButtonUpdater(){
  // Only need to update no need to re-render
  const setMessagingAtomCount = useSetRecoilState(messagingAtom);
  return <button onClick={()=>{
    // here we only need the function to set the value of
messagingAtomCount we not need the value itself if we define in this way.
    // Now for example there is any-change in the value of this state
variable then it will not result in the re-render of this component
    setMessagingAtomCount(c=>c+1);
  }}>
    Me
  </button>
}

export default App
```



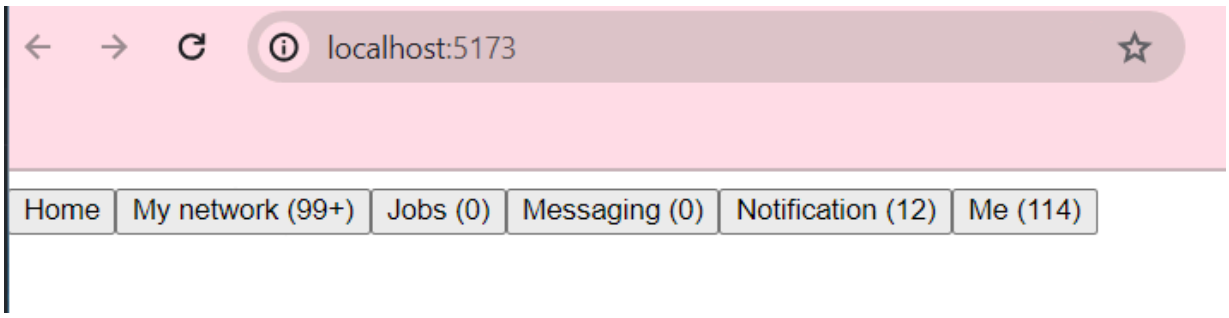
We will use the existing four values to get the final no. It is derived from the four atoms value.

(**Hard Coded Values**)

Method 1:

```
import { useRecoilValue,RecoilRoot } from "recoil"
import { jobsAtom, messagingAtom, networkAtom, notificationsAtom } from
"./atom"
// RecoilRoot , Every component that uses Recoil need to be wrapped under
it

function App(){
  return <RecoilRoot>
    <MainApp />
  </RecoilRoot>
}

function MainApp() {
  const networkNotificationCount = useRecoilValue(networkAtom)
  const jobsAtomCount = useRecoilValue(jobsAtom);
  const notificationsAtomCount = useRecoilValue(notificationsAtom);
  const messagingAtomCount = useRecoilValue(messagingAtom)
  const totalNotificationCount = networkNotificationCount+
jobsAtomCount+notificationsAtomCount+messagingAtomCount;
  // gives two thing ,actual value and way to update that value.

  return (
    <>
        <button>Home</button>

        <button>My network ({networkNotificationCount >=
100?"99+":networkNotificationCount})</button>
        <button>Jobs ({jobsAtomCount >= 100?"99+":jobsAtomCount})</button>
        <button>Messaging ({messagingAtomCount >=
100?"99+":messagingAtomCount})</button>
```

```
        <button>Notification ({notificationsAtomCount >=
100?"99+":notificationsAtomCount})</button>

        <button>Me ({totalNotificationCount})</button>

    </>
  )
}
export default App
```



Method 2 : useMemo()

```
  const totalNotificationCount = useMemo(() =>{
    return
networkNotificationCount+jobsAtomCount+notificationsAtomCount+messagingAto
mCount;
  },[networkNotificationCount, jobsAtomCount, notificationsAtomCount,
messagingAtomCount])
```

Method 3: selector()

Selector is derived from other atoms/selectors

Selector consist of key and get (where get is a function which give us access to the get props)

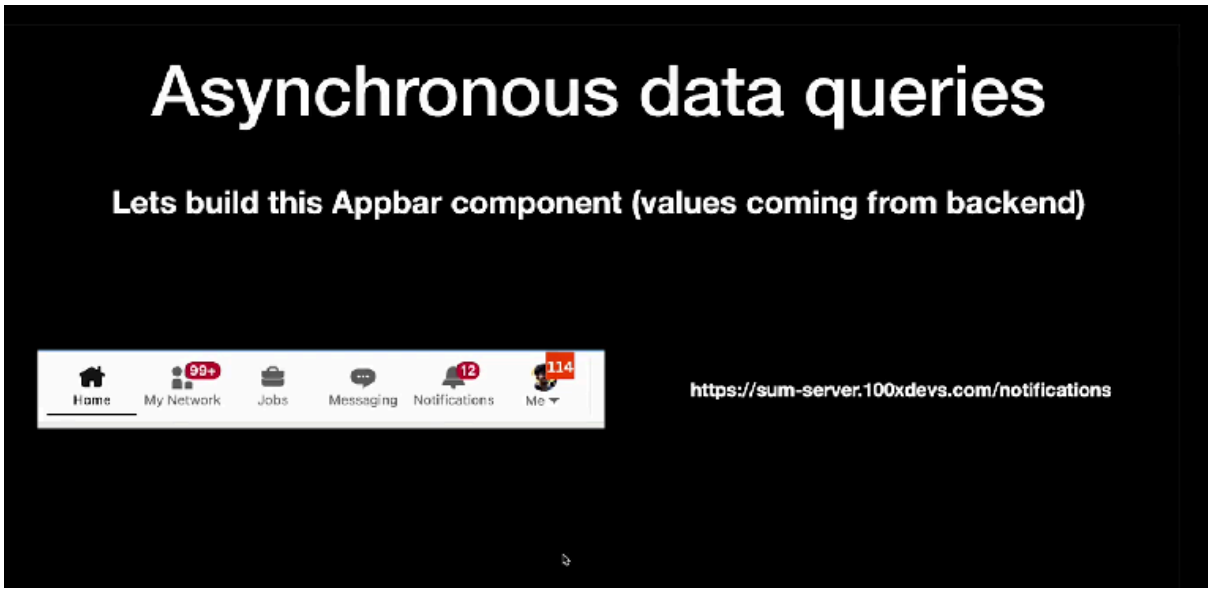
```
export const totalNotificationSelector = selector({
```

```
    key: "totalNotificationSelector",
    value: ({get}) => {
        const networkAtomCount = get(networkAtom);
        const jobsAtomCount = get(jobsAtom);
        const notificationsAtomCount = get(notificationsAtom);
        const messagingAtomCount = get(messagingAtom);
        return
networkAtomCount+jobsAtomCount+notificationsAtomCount+messagingAtomCount
    }
})
```

App.jsx

```
const totalNotificationCount = useRecoilValue(totalNotificationSelector);
```

Async Data Queries



Starter Code:

```
import './App.css'
import { RecoilRoot, useRecoilState, useRecoilValue, useSetRecoilState }
from 'recoil'
import { notifications, totalNotificationSelector } from './atoms'
import { useEffect } from 'react'
```

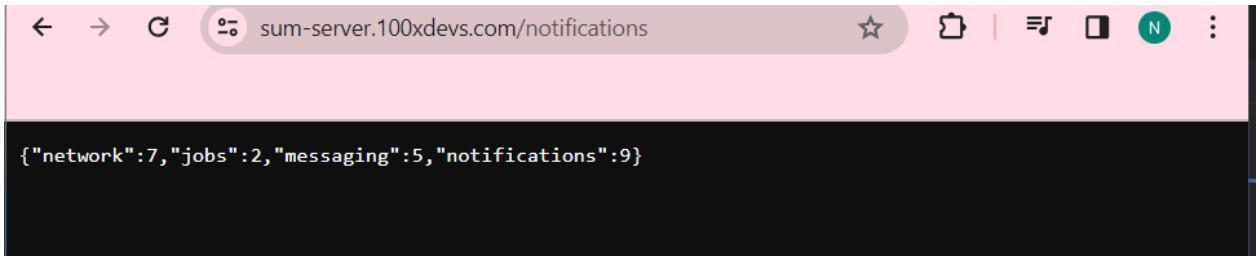
```
import axios from 'axios'

function App() {
  return <RecoilRoot>
    <MainApp />
  </RecoilRoot>
}

function MainApp() {
  const [networkCount, setNetworkCount] = useRecoilState(notifications)
  const totalNotificationCount =
useRecoilValue(totalNotificationSelector);

  useEffect(() => {
    // fetch
    axios.get("https://sum-server.100xdevs.com/notifications")
      .then(res => {
        setNetworkCount(res.data)
      })
  }, [])

  return (
    <>
      <button>Home</button>

      <button>My network ({networkCount.networks >= 100 ? "99+" :
networkCount.networks})</button>
      <button>Jobs {networkCount.jobs}</button>
      <button>Messaging ({networkCount.messaging})</button>
      <button>Notifications ({networkCount.notifications})</button>

      <button>Me ({totalNotificationCount})</button>
    </>
  )
}

export default App
```

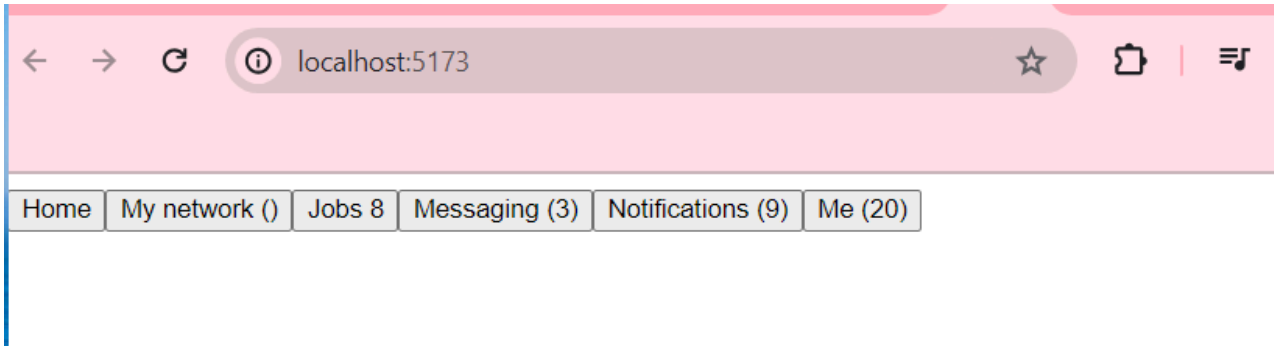
Atom.jsx

Consolidated the previous four atoms into a single atom.

```
import { atom, selector } from "recoil";

export const notifications = atom({
    key: "networkAtom",
    // storing all in single atom.
    default: {
        network: 4,
        jobs: 6,
        messaging: 3,
        notifications: 3
    }
});

export const totalNotificationSelector = selector({
    key: "totalNotificationSelector",
    get: ({get}) => {
        const allNotifications = get(notifications);
        return allNotifications.network +
        allNotifications.jobs +
        allNotifications.notifications +
        allNotifications.messaging
    }
})
```

Our backend look like this





It will start show the default values and then when it update data from the backend call then it will update, we will see kind of refreshing of our page.

This approach is not **good./ right way**
First we see a flash when response comes.
We should;dnt default them to zero we kind of want to move useEffect inside atom.

We may think that this may be the right approach but it isn't

```
export const notifications = atom({
    key: "networkAtom",
    // storing all in single atom.
    default: async() => {
        const res = await
axios.get("https://sum-server.100xdevs.com/notifications")
        return res.data;
    }
});
```

If we know that default value of the atom is coming asynchronously then we will define in this way
default: is a selector itself

```
import { atom, selector } from "recoil";
import axios from "axios"

export const notifications = atom({
    key: "networkAtom",
    // storing all in single atom.
    default:selector({
        key:"networkAtomSelector",
        get:async() => {
            const res = await
axios.get("https://sum-server.100xdevs.com/notifications")
            return res.data;
        }
    })
});
```

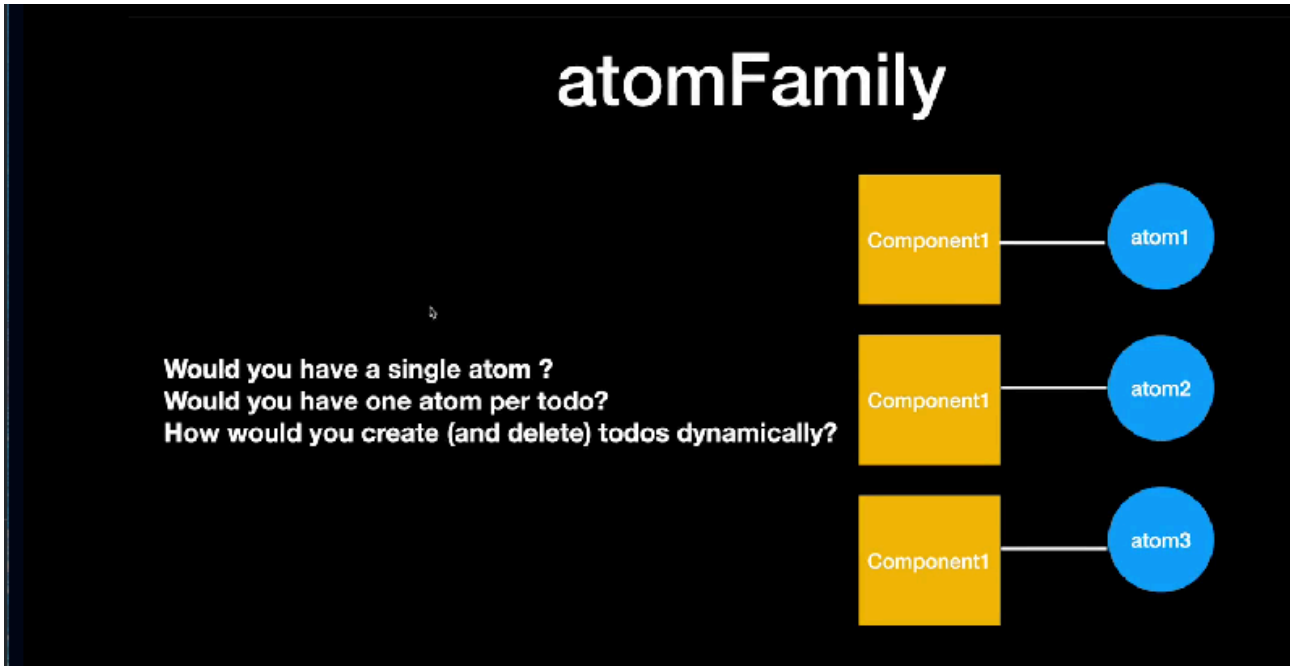
In this nothing re-renders till we get the data .

How to put a loader till we get data from backend

atomFamily

Problem: Sometimes you need more than one atom for your usecase.

For example: Creating a todo application.



The problem is each component, needs to create its own atom, we have restriction that we can't define all in a single atom.
If id is the same it needs to hit the same atom.

The solution is atom family,rather than subscribing to the atoms we will subscribe to the atomFamily.

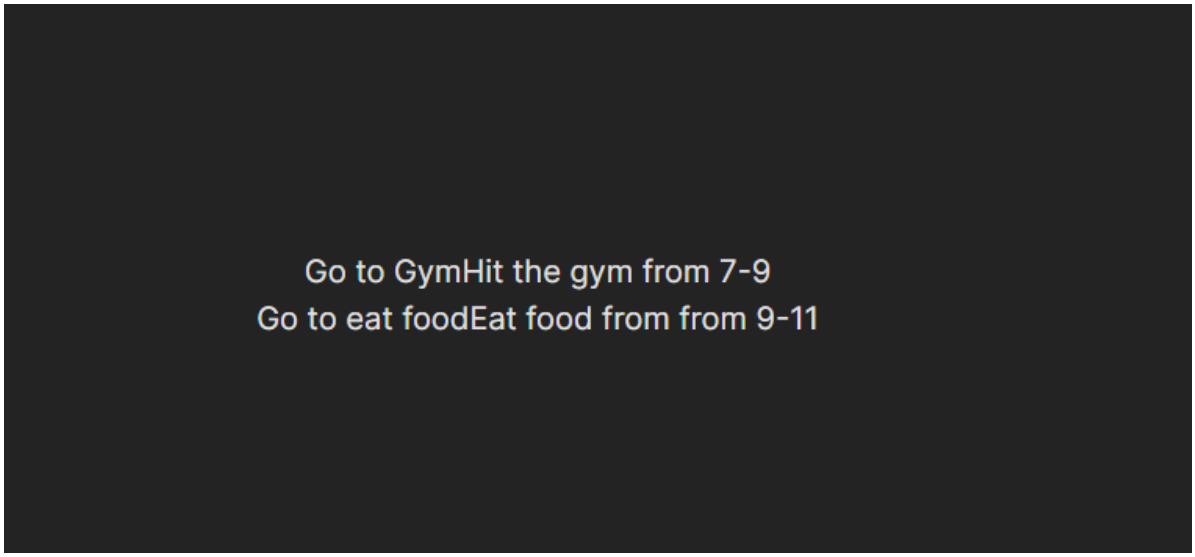
```
import { atomFamily } from "recoil";
import { TODOS } from "./todos";

// way to create multiple atoms
export const todosAtomFamily = atomFamily({
  key: 'todosAtomFamily',
  // default value is no more value it is now a function
  default: id => {
    return TODOS.find(x => x.id === id)
    // every todo need to have an atom associated with t hem
  }
```

```
  // default:id => {
  //   let foundTodo = null;
  //   for(let i =0; i<TODOS.length; i++){
  //     if(TODOS[i].id === id){
  //       foundTodo = TODOS[i];
  //     }
  //   }
  // return foundTodo
  // }
});

// const todoAtom = atom({
//   key:"todoAtom",
//   default:1
// })
```

App.jsx

```
import './App.css'
import { atom,RecoilRoot, useRecoilState, useRecoilValue } from 'recoil';
import { todosAtomFamily } from './atoms';

function App() {
  return <RecoilRoot>
    <Todo id={1}/>
    <Todo id={2} />
  </RecoilRoot>
}

function Todo({id}) {
  const currentTodo = useRecoilValue(todosAtomFamily(id))
  // i just want atom one from this specific todo id from the todo family
  return (
    <>
      {currentTodo.title}
      {currentTodo.description}
      <br />
    </>
  )
```

```
}

export default App
```



We can define many atom but we don't know what todo will come , hence we will need to make a function to create a Atom

```
function giveAtom(){
        return atom();
}
```

We may think that we can put all the values of TODO in a single atom like this:

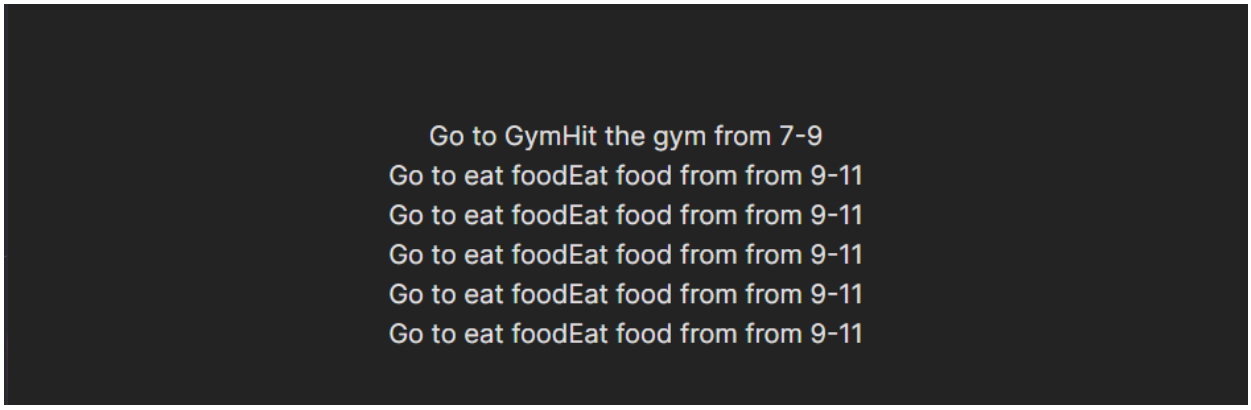
```
export const todosAtom = atom({
        key: "atom",
        default: TODOS
})
```

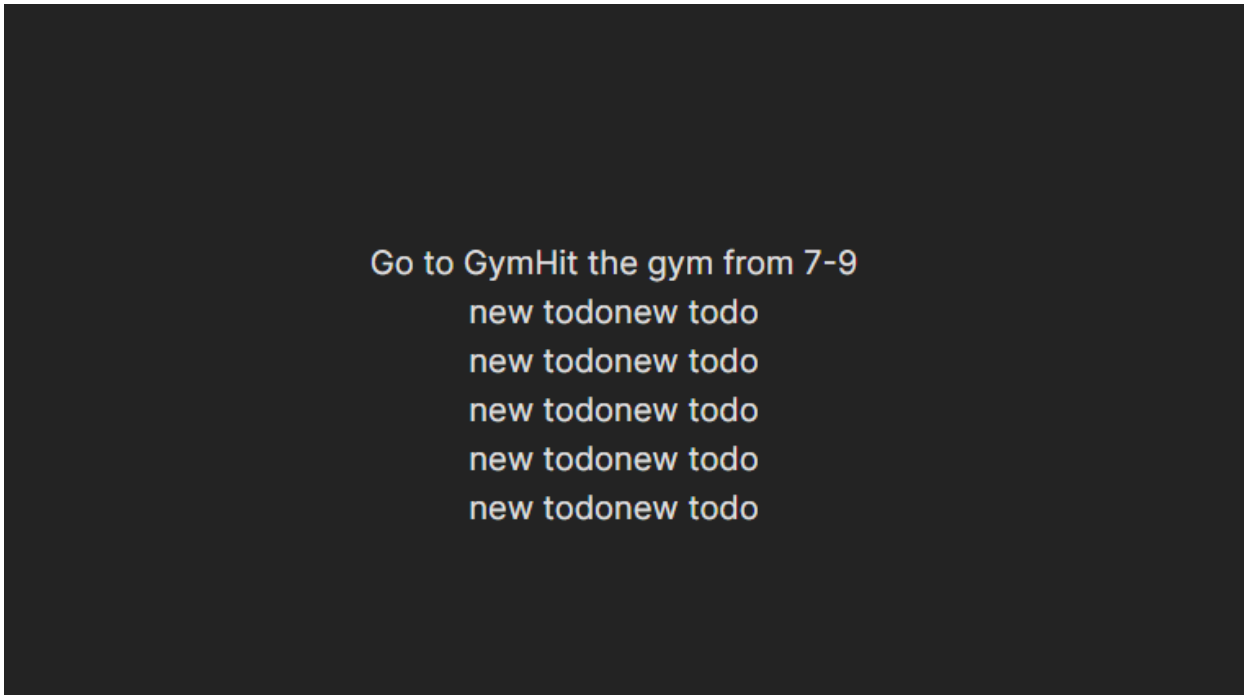
Any change in even a single todo will result in re-rendering of the entire component.

If we use AtomFamily approach suppose todo1 changes then only todo1 component will re-render else will not.

AtomFamily stores a function which returns a new atom to us.
Basically dynamically create atoms

Lets update todosAtomFamily with id =2 , and see how it affect.






```
function App() {
  return <RecoilRoot>
    <UpdaterComponent />
```

```
    <Todo id={1}/>
    <Todo id={2} />
    <Todo id={2} />
    <Todo id={2} />
    <Todo id={2} />
    <Todo id={2} />
  </RecoilRoot>
}

function UpdaterComponent(){
  const updateTodo = useSetRecoilState(todosAtomFamily(2));
  useEffect(()=>{
    setTimeout(()=>{
      updateTodo({
        id:"2",
        title:"new todo",
        description:"new todo"
      })
    }, 5000)
  }, [])
}
```

selectorFamily

In the TODO application, lets say you are supposed to get TODOs from a server.

https://sum-server.100xdevs.com/todo?id=1

Default value of the atomFamily cannot be asynchronous but default value of selectorFamily can be asynchronous.

Function returns a function

Watch Video again!!!

useRecoilStateLoadable, useRecoilValueLoadable