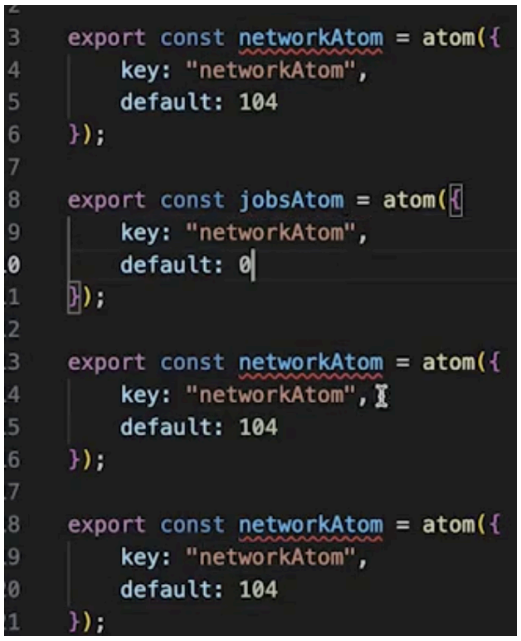
11.3 - Recoil Deep Dive

- You can have multiple export statements in the same file, or you can write their export at the end of the file:



```
export const networkAtom = atom({
    key: "networkAtom",
    default: 104
});

export const jobsAtom = atom({
    key: "networkAtom",
    default: 0
});

export const networkAtom = atom({
    key: "networkAtom",
    default: 104
});

export const networkAtom = atom({
    key: "networkAtom",
    default: 104
});
```

- When you want the default value of an atom to come from an asynchronous backend call (like fetching user's data), you cannot set the default to an async function directly. So you instead set it to a selector (because a selector is allowed to be asynchronous) which calls the asynchronous function and returns the value from the backend.
    - Yes, an atom's default value can be a selector.

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

- When you have to create an atom per component (like one atom for one todo component and there are multiple todos, thus the need for one atom for each todo), this is when you use the atom family.
    - Syntax: const aTodo = useRecoilValue(todo(1));
        - Pass the atom id you want from the todo atom family as a parameter.

    const todoFamily = atomFamily({

        key: 'atomFamily',

        default: (id) => {return TODOS.find(x => x.id === id)}

    }

- A use case of atomFamily, storing each individual post which is the same component but different atom per post.

- AtomFamily is basically a function which returns a new atom to you (or a cached one if it was previously created).

```
13  export const todosAtomFamily = atomFamily({
14    key: 'todosAtomFamily',
15    default: id => {
16      return TODOS.find(x => x.id === id)
17    },
18  });
```

- You need to use selectorFamily inside of atomFamily because even the selector is dynamic (it returns a different ‘todo’ from the backend dynamically based on the id).
    - Using a selector would produce duplicate keys which is an issue.
    - The get function of the selectorFamily is a function which returns an async function which actually does the fetch from the backend (you cannot have a direct async function in the selectorFamily’s get).

```
export const todosAtomFamily = atomFamily({
  key: 'todosAtomFamily',
  default: selectorFamily({
    key: "todoSelectorFamily",
    get: function(id) {
      return async function ({get}) {
        const res = await axios.get(`https://sum-server.100xdevs.com/todo?id=${id}`);
        return res.data.todo;
      }
    }
  })
});
```

- To show a loader while you are receiving the backend call (by default it shows a blank screen if the atom hasn’t received the info from the backend yet), you can utilise **useRecoilStateLoadable**(atomFamily(id)) which returns {content, state}, where content contains your original object of the todo and state is either “loading”, “hasValue”, or “hasError”, through which based on an if condition, you can either show a loading bar (or a skeleton loading bar) or if hasValue, then render the todo normally (or if hasError, then render an error component).
    - Or if you just want the value and not setter function, then utilise useRecoilValueLoadable().
-