Bootcamp de Desarrollo Web

Sprint 4

Temario

Temario

- Redux Toolkit (continuación).
  - `createReducer`
  - `createAction`
  - `createSlice`

createAction

createAction

Es una función que simplifica la creación de *actions*.

Forma “tradicional” de crear *actions*:

```
const INCREMENT = "counter/increment";

function increment(amount) {
  return {
    type: INCREMENT,
    payload: amount,
  };
}

const action = increment(3);
```

Crear una *action* con RTK:

```
import { createAction }
  from "@reduxjs/toolkit";

const increment =
  createAction("counter/increment");

const action = increment(3);
```

El argumento recibido se convierte en el payload de la *action*.

createReducer

`createReducer` (1/3)

Es una función que simplifica la creación de *reducers*.

Internamente utiliza una librería llamada Immer que facilita (drásticamente) la lógica de actualizaciones inmutables. Gracias a Immer el desarrollador podrá escribir código “mutable” dentro de los *reducers*. 🤯

`createReducer` (2/3)

Forma “tradicional” de crear *reducers*:

```
const initialState = { value: 0 };

function counterReducer(state = initialState, action) {
  switch (action.type) {
    case "increment":
      return { ...state, value: state.value + 1 };
    case "decrement":
      return { ...state, value: state.value - 1 };
    case "incrementByAmount":
      return { ...state, value: state.value + action.payload };
    default:
      return state;
  }
}
```

Crear un *reducer* con RTK:

```
import { createAction, createReducer } from "@reduxjs/toolkit";
const increment = createAction("counter/increment");
const decrement = createAction("counter/decrement");
const incrementByAmount = createAction("counter/incrementByAmount");
const initialState = { value: 0 };

const counterReducer = createReducer(initialState, (builder) => {
  builder
    .addCase(increment, (state, action) => {
      state.value++;
    })
    .addCase(decrement, (state, action) => {
      state.value--;
    })
    .addCase(incrementByAmount, (state, action) => {
      state.value += action.payload;
    });
});
```

`createSlice`

Alternativa a utilizar `createAction` y `createReducer` por separado.

`createSlice` (1/4)

Típicamente, el estado en Redux se organiza en *"slices"* (rebanadas), definidas por los *reducers* que se pasan a `configureStore` (o a `combineReducers`).

```
import { configureStore } from "@reduxjs/toolkit";
import tasksReducer from "./tasksReducer";
import usersReducer from "./usersReducer";

const store = configureStore({
  reducer: {
    tasks: tasksReducer,
    users: usersReducer
  },
});
```

En este ejemplo, tanto `tasks` como `users` son considerados *"slices"*. Cada uno de estos *reducers*:

- Es "dueño" de una parte del estado.
- Define cómo el estado se actualiza.
- Especifica qué *actions* generan cambios en el estado.

`createSlice` (2/4)

Para simplificar el proceso de crear de crear *reducers*, *action types* y *action creators*, RTK incluye la función `createSlice` 🤯. Todo esto se crea en base de los nombres que se usaron para crear al *reducer*. Internamente se está usando `createAction` y `createReducer`.

```
const tasksSlice = createSlice({
  name: "tasks",
  initialState: [],
  reducers: {
    createTask(state, action) {...},
    updateTask(state, action) {...},
    deleteTask(state, action) {...},
  },
});
```


```
{
  name: "tasks",
  actions: {
    createTask,
    updateTask,
    deleteTask,
  },
  reducer,
}
```

`createSlice` (3/4)

`createSlice` toma las funciones definidas en el atributo *reducers* y por cada *"case reducer"* provisto se genera un *action creator* que utiliza el nombre del *reducer* como *action type*.

Ejemplo: el *reducer* `createTask` genera el *action type* llamado "`tasks/createTask`" y el `createTask()` *action creator* retornará una acción con dicho tipo.

```
const { createTask } = tasksSlice.actions;

createTask({ id: 123, name: "Estudiar React y Redux" });
```

`createSlice` (4/4)

En general, cuando se define un *slice*, se querrá exportar los *actions creators* y el *reducer*. Pero también se podría haber exportado únicamente el *slice*.

```
const tasksSlice = createSlice({
  name: "tasks",
  initialState: [],
  reducers: {
    createTask(state, action) {...},
    updateTask(state, action) {...},
    deleteTask(state, action) {...},
  },
})

const { actions, reducer } = tasksSlice;
export const { createTask, updateTask, deleteTask } = actions;
export default reducer;
```