Redux 1.2_CW Exercises

actions & action creators

ex01: create a reducer for todo list

challenge

Imagine you have a list of todos that you want to manage in your app's state. You need a reducer to define how your state changes when you add or remove a todo.

Create a `todosReducer` that handles adding and removing todos in `todosReducer.js`.

https://codesandbox.io/s/rx1-2-cw-ex01-qkgmkq

solution

```
const initialState = { todos: [] }

const todosReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'todos/added':
      return { ...state, todos: [...state.todos, action.payload] }
    case 'todos/removed':
      return {
        ...state,
        todos: state.todos.filter((_, index) => index !== action.payload),
      }
    default:
      return state
  }
}

export default todosReducer
```

COPY

ex02: create a store for todo list

challenge

Just like with cookies, you need a store to manage the state of your todo list. The store keeps track of the state changes, which are managed by the reducer.

In `index.js`, import the required `createStore` function from Redux and `**todosReducer**`.

Create a Redux store by passing your `**todosReducer**` to the `**createStore**` function.

https://codesandbox.io/s/rx1-2-cw-ex02-nflkfm

solution

```
import { createStore } from 'redux'
import todosReducer from './todosReducer'

const store = createStore(todosReducer)
```

COPY

ex03: subscribe to the store for todo list

challenge

Subscribing to the store allows you to listen for state changes. Whenever a todo is added or removed, the subscription function is called, and you can respond to the state updates.

Use the `subscribe` method of the Redux store to listen for state changes and log the state.

https://codesandbox.io/s/rx1-2-cw-ex03-lhcgl6

solution

```
store.subscribe(() => console.log(store.getState()))
```

COPY

ex04: create actions and action creators

understanding

Actions are plain JavaScript objects that describe what happened in your app. Action creators are functions that create these action objects. This helps keep your code organized and consistent.

https://codesandbox.io/s/rx1-2-cw-ex04-vmvy3f

challenge

1. Create a `actions.js` and define two constants: `ADD_TODO` and `REMOVE_TODO`.
2. Create an action creator function named `addTodo` that takes a `text` parameter and returns an action object with the `type` of `ADD_TODO` and the `payload` as `text`.
3. Create an action creator function named `removeTodo` that takes an `index` parameter and returns an action object with the `type` of `REMOVE_TODO` and the `payload` as `index`.

solution

```
export const ADD_TODO = 'todos/added'
export const REMOVE_TODO = 'todos/removed'

export const addTodo = (text) => ({
  type: ADD_TODO,
  payload: text,
})

export const removeTodo = (index) => ({
  type: REMOVE_TODO,
  payload: index,
})
```

COPY

ex05: update the todosReducer

challenge

https://codesandbox.io/s/rx1-2-cw-ex05-kzld44

solution

```
import { ADD_TODO, REMOVE_TODO } from './actions'

const initialState = { todos: [] }

const todosReducer = (state = initialState, action) => {
  switch (action.type) {
    case ADD_TODO:
      return { ...state, todos: [...state.todos, action.payload] }
    case REMOVE_TODO:
      return {
        ...state,
        todos: state.todos.filter((_, index) => index !== action.payload),
      }
    default:
      return state
  }
}

export default todosReducer
```

COPY

ex06: interacting with the todo list

challenge

`index.html` contains an input field for entering a new todo, an "Add Todo" button, a "Remove Todo" button for each todo, and a list displaying the todos. Your task is to write JavaScript code that interacts with these elements using `document.getElementById`.

1. Inside the `index.js` file, create constants using the `document.getElementById` method to select each of these elements by their respective ids. Name the constants as follows:
    - The input field: `todoInput`
    - The "Add Todo" button: `addButton`
    - The list element: `todoList`

https://codesandbox.io/s/rx1-2-cw-ex06-ptfwqz

solution

```
const addButton = document.getElementById('add')
const todoInput = document.getElementById('todo-input')
const todoList = document.getElementById('todo-list')
```

COPY

ex07: updating the todo list

challenge

https://codesandbox.io/s/rx1-2-cw-ex07-4t8ctm

Implement event handlers that will dispatch actions when interacting with the "Add Todo" and "Remove Todo" buttons.

1. Create a function named `addTodoHandler`. Inside this function:
    - Get the value from the input field using the `value` property of the `todoInput` constant.
    - Check if the `text` variable is not empty.
    - If the `text` variable is not empty, dispatch the `addTodo` function with the `text`
    - Finally, reset the value of the input field to an empty string.
2. Create a function named `removeTodoHandler`. This function should be attached to the `window` object so that it can be accessed globally.

- The function should take an `index` as a parameter, which will represent the index of the todo item to be removed.
- Inside the `removeTodoHandler` function, dispatch the `**removeTodo**` function with the `index`.

3. Attach event listener to the "Add Todo" button. Call the corresponding handler function when the button is clicked. We will call the `removeTodoHandler` function when we will display the todos.

understanding

Attaching the `removeTodoHandler` function to the `window` object allows it to be accessed from outside the current scope. In our case, it's likely that the function is being used as an event handler within the HTML content, which is not within the scope of the current JavaScript file.

By attaching it to the `window` object, you make sure that it can be called as a global function. This way, when the "Remove Todo" button is clicked in the HTML content, it can find and execute the `removeTodoHandler` function globally.

solution

```
const addTodoHandler = () => {
  const text = todoInput.value
  if (text) {
    store.dispatch(addTodo(text))
    todoInput.value = ''
  }
}

window.removeTodoHandler = (index) => {
  store.dispatch(removeTodo(index))
}

addButton.addEventListener('click', addTodoHandler)
```

COPY

ex08: displaying the todo list

challenge

https://codesandbox.io/s/rx1-2-cw-ex08-z6t28y

Your task is to write JavaScript code that will display the list of todos and update it whenever the state changes.

1. Create a function named `updateTodoList` that will be responsible for updating the displayed list of todos on the webpage. Inside this function, get the current state from the Redux store using `store.getState()`, and update the content of the `todoList` element with the todos from the

state. For each todo, include a "Remove Todo" button that calls the `removeTodoHandler` function with the corresponding index.

2. Inside the `store.subscribe` method, after the log, call the function `updateTodoList`.
3. Call the `updateTodoList` function after defining it to ensure that the initial list of todos is displayed correctly on page load.

solution

```
store.subscribe(() => {
  console.log(store.getState())
  updateTodoList()
})

const updateTodoList = () => {
  const state = store.getState()
  todoList.innerHTML = state.todos
    .map((todo, index) => {
      return `<li>${todo} <button onclick="removeTodoHandler(${index})">Remove</button></li
    })
    .join('')
}

updateTodoList()
```

COPY

entire solution

https://codesandbox.io/s/rx1-2-cw-entire-solution-dmg97f