Redux

By: Mosh Hamedani

Avoiding Array Mutations

In this app, we're working with an array of items (todos), and avoiding the mutation of this array is the challenging part of this exercise. But with the following tips, you'll realise it's not as challenging as you might think.

Let's imagine we have the following array of complex objects:

```
var source = [ {…}, {…}, {…} ];
```

Adding an item

Instead of the **push()** method, we use the **concat()** method to add an item to an array, because the **push()** method mutates the original array, whereas **concat()** returns a new array:

```
var target = source.concat(newItem);
```

Removing an item

We use the **filter()** method to create a new array that excludes the given item:

```
var target = source.filter(i => i.id !== id);
```

Now, **target** includes all items in the source except the one with the given id.

Updating an item

We have to find the copy all items before and after the given item to the new array, and use **tassign()** to get a copy of this item and apply mutations before placing it in the target at the right position.

So, first we need to find the position of this item in the array:

```
var item = source.find(i => i.id === id);
var index = source.indexOf(item);
```

Now, to get all the items before and after this item, we use the **slice()** method:

```
var beforeItems = source.slice(0, index);
var afterItems = source.slice(index + 1);
```

Now, we take a copy of the given item and apply mutations using **tassign()**:

```
var updatedItem = tassign(item, { … });
```

Finally, we need to put all these items in a new array. One way to do this is:

```
var newArray =
[].concat(beforeItems).concat(updatedItem).concat(afterItems);
```

But this is too verbose. We can use the spread operator (…) to enumerate items in an array:

```
var newArray = [...beforeItems, updatedItem, ...afterItems];
```