Exercise 2 - TDD in a framework

Description

This time, we'll take a look at using TDD in a specific framework. It's slightly different for each one, but in this case, we'll take a look at Vue and VueX. More specifically, we'll be testing the effect on our *store* by dispatching *actions*.

We'll walk you through setting up the first test-case all the way up to a passing test. After that, you can apply the same idea to the rest of the requirements.

Steps

- Enter the folder for `exercise-2` and run `npm i` in it.
- Start the application by running `npm start`.
- Point your browser to `http://localhost:9000/`. You should see a small TODO-list.
- The application is wired up using a VueX store. All actions are already specified and bound, but they don't do anything yet. Take a quick look at the code to get acquainted with it, especially the view (`src/components/TodoList.vue`) and the store (`src/store/modules/list.js`)
- Open the file `test/list-tests.js`.
- It should be possible to add new items by typing a text in the textbox at the bottom of the website. We'll be writing `asynchronous` tests, since we're dealing with actions. Add the following inside the `describe` in the test file:

```
it('should update the text for the next item', async () => {
    const store = createStore();
    await store.dispatch("onUpdateNext", "next item");
    assert.strictEqual(store.state.list.next, "next item");
});
```

- Note that we're always `await`ing the dispatch.
- Run the tests using `npm t`. It should fail.
- Make the test pass by going to `list.js` and finding the function `onUpdateNext`. Add a `commit` in this function. For instance:

```
commit('updateNext', payload);
```

- The exact name of the mutation is up to you, just make sure it's the same in the next step.

- Create a function `updateNext` (or whatever name you gave it) in the `mutations` object, looking like this:

```
updateNext(state, payload) {
    state.next = payload;
}
```

- Run your tests again with `npm t`. Your test should pass now.

Following the same line of steps/reasoning as we just did, use TDD to implement the following requirements. Don't forget to run your tests between each cycle, plus make sure it works in the actual website too after getting tests to pass.

- When clicking add (dispatches a `onAddNewItem`), it should add a new item to the list (the list is called `items` in the state). Each item should have a `checked` property initially set to false and a `text` property, set to whatever `next` in the state happens to be when pressing add.
- When clicking add, the `next` text should be cleared.
- Clicking the checkbox on an item triggers a `onToggleItem` dispatch, passing along the index in the `items` array. Make sure that the `checked` property for the correct item is toggled from `true` to `false` and vice versa when clicking a checkbox in the list.
- Add an `error` property to the state, which is initially an empty string. Render it somewhere in the view by adding something along the lines of:

```
<div>{{list.error}}</div>
```

- The list should not allow duplicate strings. Make sure the error message is non-empty if you do (and that no new items were added)
- The list should only allow adding 10 items at a time. Make sure the error message is non-empty if you try to (and that no more than 10 items were ever added)
- The error message should always be cleared after adding a new item successfully.

Stretch task

Add a button that lets you *remove* all items that are checked. Don't forget to add tests first!