Advanced actions

Let's start out by adding a new capability to our app. When the user clicks/taps a film on the main page, we'll tell them which one they selected. We'll do this through Redux by dispatching an action.

1. Edit App.js and change the JSX. Add in the highlighted parts below:

```
<section>
  {state.films.map(film => <div
    onClick={_=>dispatch({ type: "SET_CURENT_FILM" })}
    key={film.id}>{film.title} - {film.tagline}</div>)}
  <h1>You chose {state.currentFilm?.title}</h1>
</section>
```

2. Run and test. If you typed in exactly what we had above, the code will run but nothing happens. We purposely introduced two easy-to-make dispatch errors that we will now fix.

Adding action types

The first problem was that SET_CURRENT_FILM is misspelled, a very easy mistake to make. Let's see if we can make it a little easier to not do.

3. Create a new file in the store folder called actions.js.
4. export an actionTypes object kind of like this:

```
export const actionTypes = {
  SET_CURRENT_FILM: "SET_CURRENT_FILM",
  SET_FILMS: "SET_FILMS",
};
```

5. Now edit App.js and import this actionTypes const at the top.
6. Then find your store.dispatch and change it to ...

```
dispatch({ type: actionTypes.SET_CURRENT_FILM })
```

Now that doesn't change any functionality, but it does make it slightly tougher for developers to make a mistake because as they type "actionTypes.", intellisense shows them all of the options they can choose from. Nothing is misspelled.

7. Go ahead and add in all of the other action types that are currently in reducer.js.
8. Bonus!! Open reducer.js, import actionTypes at the top and replace all of the case tests from strings like *"SET_FILMS"* to actionTypes like *actionTypes.SET_FILMS*.

Adding the action creators enumeration

The other problem we introduced was that SET_CURRENT_FILM needs a film to set and we 'forgot' to add it. This is another typical mistake that developers make because we forget what is needed in the payload. To solve it, we expose an action creator that will prompt the user with all arguments needed for a given reducer action.

9. Add this to actions.js:

```
const setCurrentFilm = film => ({type: actionTypes.SET_CURRENT_FILM, film});
export const actions = {
  setCurrentFilm,
```

```
};
```
This merely exports an *actions* object that has a setCurrentFilm method. setCurrentFilm receives a *film* and returns an object with a properly-spelled type and a properly-named argument (film). This object is a perfectly-formed action that can then be dispatched.

10. Edit App.js. Import *actions* at the top of the file.
11. Then change your dispatch to look like this:

```
dispatch(actions.setCurrentFilm(film))
```

Do you see it now? Rather than having to remember how the action object for a SET_CURRENT_FILM dispatch should look, we're creating it automatically and making it more difficult for a developer to get wrong.

12. Go ahead and add action creators for all of the cases in reducer.js.
13. Lastly, edit App.js. In useEffect, there's a dispatch to a SET_FILMS action. Refactor that to use one of your action creators.