The Actions

This lesson introduces actions and their creation in the Flux architecture

WE'LL COVER THE FOLLOWING

- The actions

The actions

You probably noticed that we didn't talk about the actions. What are they? The convention is that they should be simple objects having two properties - `type` and `payload`:

```
{
  type: 'USER_LOGIN_REQUEST',
  payload: {
    username: '...',
    password: '...'
  }
}
```

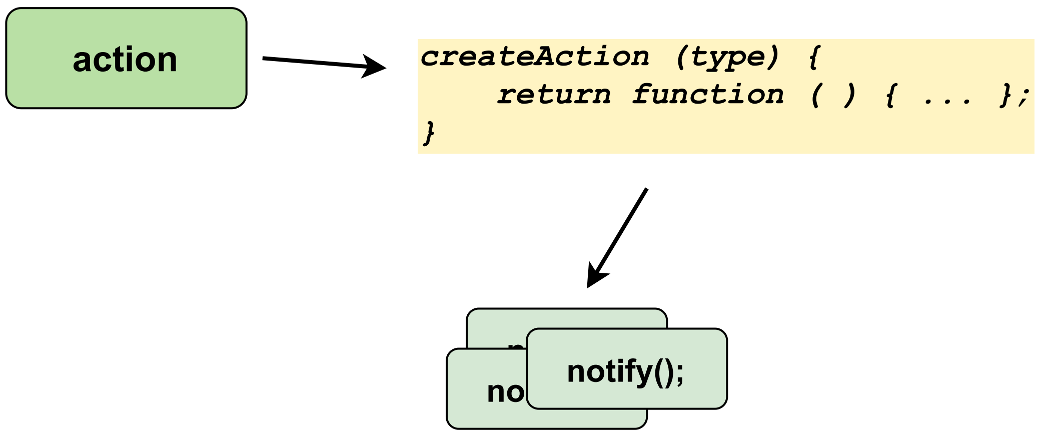
The `type` says what exactly the action is and the `payload` contains the information associated with the event. And in some cases, we may leave the `payload` empty.

It's interesting that the `type` is well known in the beginning. We know what type of actions should be floating in our app, who is dispatching them and which of the stores are interested. Thus, we can apply partial application and avoid passing the action object here and there. For example:

```
var createAction = function (type) {
  // actions must have a type
  if (!type) {
    throw new Error('Please, provide action\'s type.');
  } else {
    return function (payload) {
      // pass action to dispatcher
      return dispatcher.dispatch({
        type: type,
        payload: payload
      });
    }
  }
}
```

`createAction` leads to the following benefits:

- We need not to remember the exact type of the action. We now have a function which we call passing only the payload.
- We don't need an access to the dispatcher anymore which is a huge benefit. Otherwise, think about how we have to pass it to every single place where we need to dispatch an action.
- In the end, we don't have to deal with objects but with functions which is much nicer. The objects are *static* while the functions describe a *process*.



```
createAction (type) {
    return function ( ) { ... };
}
```

Flux Action Creators

This approach for creating actions is actually really popular and functions like the one above are usually called *action creators*.

In the next lesson, we will wrap up the code, presenting the unified version.