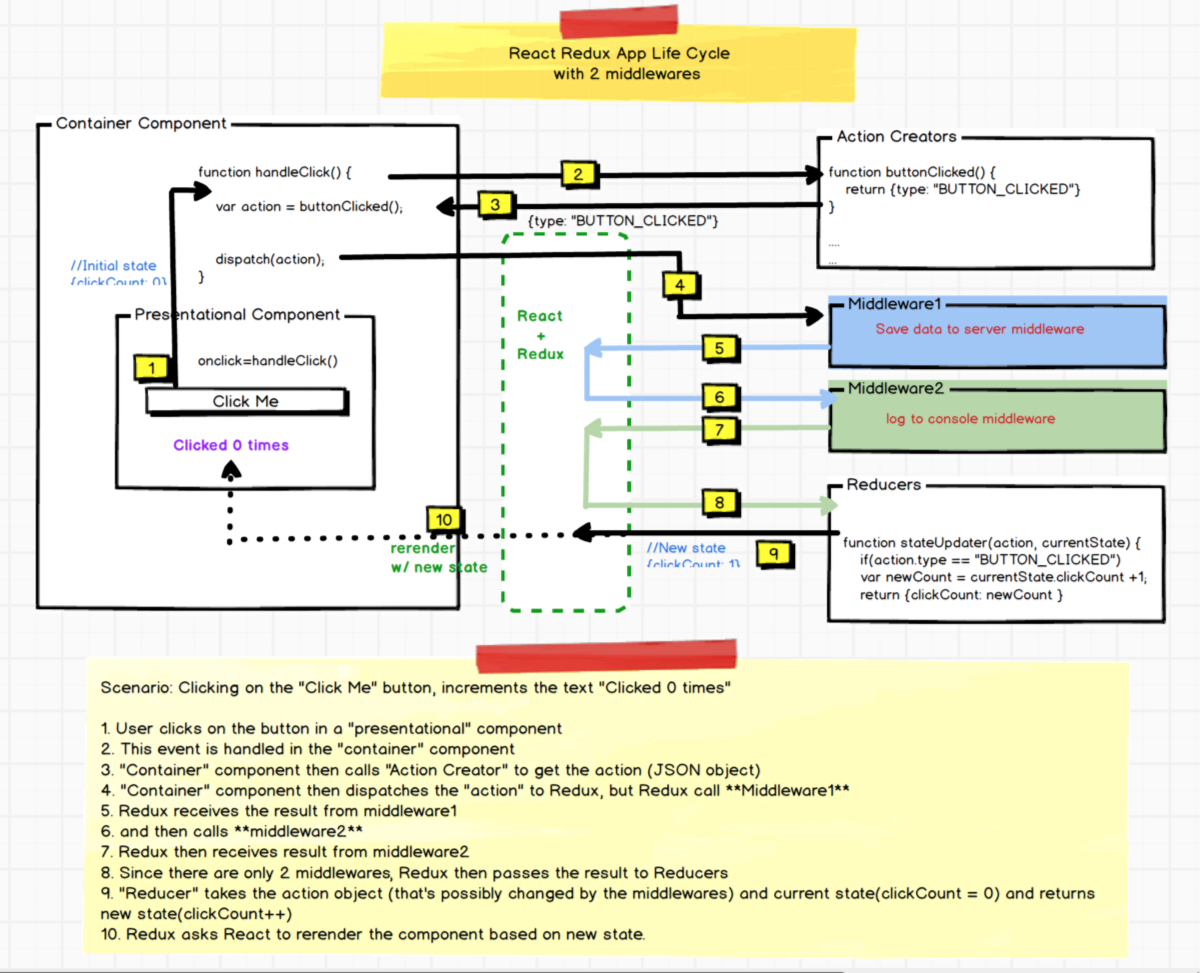


**Scenario: Now, let’s say when the user clicks the button, we want to save click count to the server and also do console logs of state changes for debugging. We could use middlewares! Let’s see how the control flow looks now.**



***In Redux, middleware functions are called one-by-one until all of them are called and then the “Action” object is sent to the “Reducers”.***

**This is to allow middlewares to potentially modify Action object, resolve AJAX calls, and do other things like logging, before finally calling reducers. Reducers will then update state and rerender the app if needed.**

we use a library called **[Axios](https://github.com/mzabriskie/axios" \t "_blank)**to make the AJAX calls**.**But Axios returns a Promise in return. **But Actions need to be pure JSON object with all the data before it reaches Reducers.**So, we ended upusing [**redux-promise**](https://github.com/acdlite/redux-promise)to intercept anytime the “Action” contains a Promise object and resolve it before the control hits the Reducers.

**How does a Middleware Work?**

*Here is the pseudo code.*

1. Redux calls middleware w/ , “dispatch”(function) and “next”(function) and the “action” JSON object.

2. Middleware inspects “action” object to see if it has a Promise.

3. If there are no Promise, then it calls “next” function to return control back to Redux.

4 If it does, then it attaches(chains) **it’s own** success and a failure callbacks and waits for the response from the server.

4.1 And once the Promise is resolved, it finally calls “dispatch” function to return result or error(from the server) and sends it back to Redux.