**ETLC\_ApexBridge**

Design pattern for communication between Lightning Components and Apex

Why do we need a design pattern to manage the communication between Lightning Components and Apex? Great question… I am glad you asked.

If you want to work with Visualforce pages, contacting the Apex controller is quite simple. You just have to define a method in Apex that returns a PageReference (to control navigation) and you write the name of this method in the Visualforce markup as the value of the action attribute. That’s It!

With Lightning Components, you have to type a bit more code… In the Apex server-side controller, you just need to annotate the method with @AuraEnabled (that part is easy) but in JavaScript you have to write at least something like this:

var action = component.get("c.getExpenses");

action.setParams({…});

action.setCallback(this, function(response) {

if (response.getState() === "SUCCESS") {

console.log(response.getReturnValue());

}

});

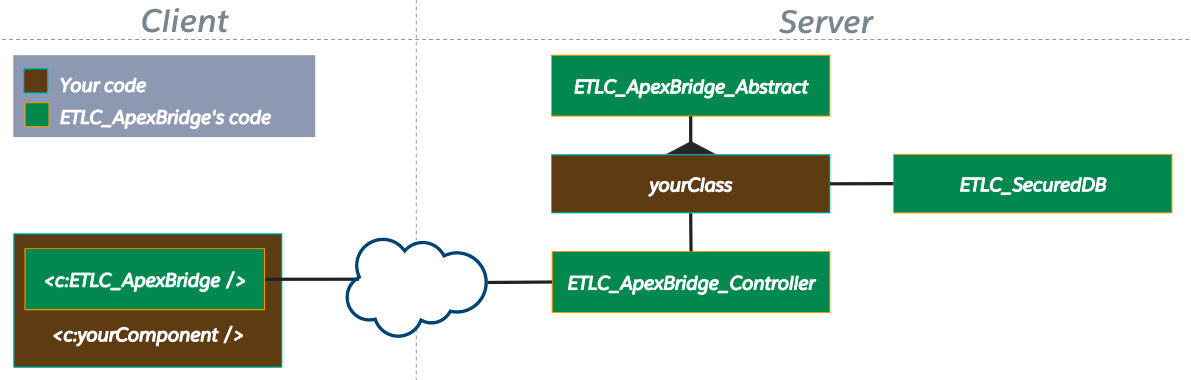
$A.enqueueAction(action);

This code gets longer if you want to properly handle errors, display a “Please Wait” message while Apex is busy, etc. and to make matters worse, this code has to be written from any component who needs to call the Apex server-side controllers!

Also, don’t forget that Lightning Components run in “system mode” and you could accidentally expose information that the users should not see. Sure, you can write Apex code that protects you from this type of accidents, but you have to write the code every time you make a SOQL or a DML… More code to write!

After a lot of thinking on how to efficiently call an Apex class from Lightning components and process the data securely in a way that is easy to implement and reusable, I came out with a design pattern that I call ApexBridge.

I have also written and polished a set of files (Lightning Components bundles and Apex classes) that implement this design pattern. I have used these files in few projects that I have worked with and I have saved about 50% development time thanks to the implementation of this design pattern. I found this library so useful, that I decided to share it with you, so I created a GitHub repository, named ETLC\_ApexBridge, that hosts the library and I wrote some documentation to help you understand how to use it in your own projects.



Let me show you the code you would need to call Apex from JavaScript using the ApexBridge design pattern. This is what the JavaScript needs to look like:

findAcccounts : function(component, helper) {

var apexBridge = component.find("ApexBridge");

apexBridge.callApex({

component: component,

request: {

controller: "ETLC\_DEMO\_03",

method: "findAccounts",

input: { howMany: 5 }

},

callBackMethod: function (request) {

component.set("v.accounts", request.output);

}

});

},

And this is what the Apex controller needs to look like:

private List<Account> findAccounts(ETLC\_ApexBridge\_Request request) {

Integer howMany = Integer.valueOf(request.getInputValue('howMany'));

return ETLC\_SecuredDB.query('SELECT ID, Name FROM Account LIMIT ' + howMany);

}

Please visit the ETLC\_ApexBridge library repository (<https://github.com/eltoroit/ETLC_ApexBridge> ) where you will find a presentation (<https://github.com/eltoroit/ETLC_ApexBridge/blob/master/ETLC_ApexBridge_UnderstandingTheDesignPattern.pps> ) that helps you understand how the design pattern works and how you can use the companion library (<https://github.com/eltoroit/ETLC_ApexBridge>). For further information, check the developer guide (<https://github.com/eltoroit/ETLC_ApexBridge/blob/master/ETLC_ApexBridge_DeveloperGuide.pdf> ) and install the sample files (<https://github.com/eltoroit/ETLC_ApexBridge_Samples> ).