Pagination JS Controller — Explanation

Detailed Explanation with Code Image

```
        // first, load total record count
        var countAction = component.get("c.getTotalContacts");
        countAction.setCallback(this, function(res){
            var total = res.getReturnValue();
            component.set("v.totalRecords", total);

            // build page list
            var totalPages = Math.ceil(total / pageSize);
            component.set("v.totalPages", totalPages);

            var pages = [];
            for (let i = 1; i <= totalPages; i++) {
                pages.push(i);
            }
            component.set("v.pageList", pages);
        });
        $A.enqueueAction(countAction);

        // then load the records for the current page
        var dataAction = component.get("c.getContacts");
        dataAction.setParams({
            pageNumber : pageNumber,
            pageSize   : pageSize
        });
        dataAction.setCallback(this, function(res){
            component.set("v.contacts", res.getReturnValue());
        });
        $A.enqueueAction(dataAction);
    }
})
```

Section 2: Expanded Explanation

This JavaScript controller function is used within an Aura component to support server-side pagination of Contact records. The function is responsible for gathering vital pagination-related information and then loading the appropriate set of records from the server so that only the data for the selected page is displayed.

Step 1: Retrieve Current Pagination Values:

The function begins by reading two component attributes: v.pageNumber (current page index) and v.pageSize (how many records are displayed per page). These values act like input parameters that allow subsequent server calls to request the correct slice of data.

Step 2: Call the Apex method 'getTotalContacts':

A server-side call is made to the Apex method getTotalContacts which returns the total number of Contacts in the system. A callback is provided so that we can handle the response once the server call completes. The returned value is saved in the component attribute v.totalRecords.

Step 3: Build Pagination Information:

Once the total number of records is known, the total number of pages is calculated by dividing the total by the pageSize and rounding up using Math.ceil. We then construct a list of all possible page indexes (1….totalPages). This list is assigned to v.pageList so the UI can display clickable page numbers. This step ensures that the user interface can navigate between pages properly.

Step 4: Enqueue the call:

The $A.enqueueAction(countAction) method is then called to instruct Aura to send the server request. Aura queues the action and processes it in the correct order.

Step 5: Load Records for the Current Page:

Another server-side action is prepared by calling the Apex method getContacts. This time we also set parameters pageNumber and pageSize so that only the records for the current page are returned. A callback is used here as well to handle the returned data.

Step 6: Store Returned Records:

Inside the callback of the getContacts action, res.getReturnValue() retrieves the list of Contact records from the server, and then these are stored inside the v.contacts attribute.

Final Result:

At the end of the function, the component will hold several important values: the total number of Contact records, a list of all page numbers, and the list of Contact records for the selected page. These values are then used by the component’s UI to render the pagination controls and populate the table of records. This approach allows efficient data loading and avoids transferring all records at once, which improves performance.