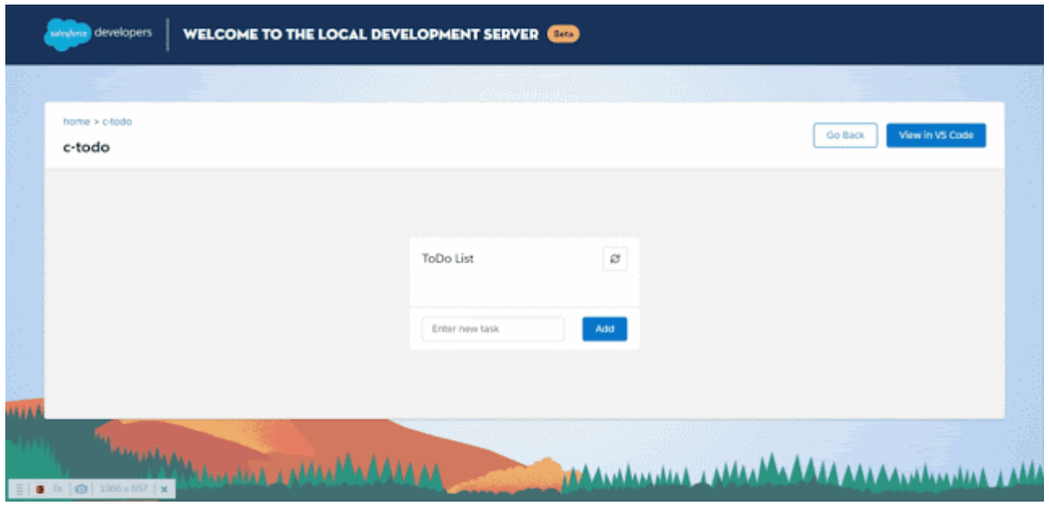
Salesforce LWC Tutorial Part 5 | ToDo App Project | Wire Service linked to Function and RefreshApex

by Rahul Malhotra *on* Wednesday, September 23, 2020 *in* Apex, JavaScript, Lightning, Lightning Basics, Lightning Tutorial, LWC, Salesforce, Salesforce Developers, ToDoApp, wire

Hello Trailblazers,

Welcome to the fifth tutorial in LWC Tutorial Series where we're building a **ToDo App Project**. We're focusing on the concept of **Learning By Doing** in this tutorial series. In this tutorial, we're going to learn how we can fetch the data from the apex controller for our ToDo List component using wire service in LWC. We're going to bind our wire service to a function in js which will receive the response. This time we'll be able to mutate the ToDoList as it won"t be directly binded with wire service. We'll also learn about RefreshApex function which will be used to refresh the browser cache.



So, let's continue building the above application. This tutorial is published on SFDC Stop YouTube Channel. So, if you want to learn in detail, have a look at the tutorial video down below. Otherwise, you can scroll down to have a look at the code gist with explanation.

Tutorial Video

Code Gist with Explanation

I highly recommend you to have a look at the video to understand in detail. However, let's have a quick look at the code below to understand in short:-

Apex Class (ToDoListController.cls)

```
/*
*       Author:- Rahul Malhotra
*       Description:- This apex class is used as a controller for our Todo List Applicati
```

```
*       Created:- 06/08/2020
*       Last Updated:- 06/08/2020
*       Code Origin:- SFDCStop (https://www.sfdcstop.com/)
*       Tutorial:- Salesforce Lightning Web Component Tutorial Series
*/
public with sharing class ToDoListController {

    /*
    *   Author:- Rahul Malhotra
    *   Description:- This method is used to return a list of tasks
    */
    @AuraEnabled(cacheable=true)
    public static List<Task> getTasks() {
        return [SELECT Subject FROM Task WHERE OwnerId =:UserInfo.getUserId()];
    }

    /*
    *   Author:- Rahul Malhotra
    *   Description:- This method is used to insert a new task in Salesforce
    */
    @AuraEnabled
    public static Task insertTask(String subject) {
        try {
            Task task = new Task(
                Subject = subject,
                OwnerId = UserInfo.getUserId(),
                Status = 'Not Started',
                Priority = 'Normal'
            );
            insert task;
            return task;
        } catch(Exception e) {
            System.debug(e.getMessage());
        }
        return null;
    }

    /*
    *   Author:- Rahul Malhotra
    *   Description:- This method is used to delete a task from Salesforce based on the r
    */
```

```
    @AuraEnabled
    public static Boolean deleteTask(Id recordId) {
        try {
            Database.delete(recordId);
            return true;
        } catch(Exception e) {
            System.debug(e.getMessage());
        }
        return false;
    }

}
```

This Gist brought to you by gist-it. force-app/main/default/classes/ToDoListController.cls view raw

As you can see above, I have created two more methods **insertTask** and **deleteTask**.

insertTask() is used to insert a new task with the subject which will be passed in this apex method as a parameter. This subject will be nothing but our todo task name that we've added to the todo list. OwnerId of the new task will be the id of logged in user, status is hardcoded as "Not Started" and the priority is "Normal". The task inserted is returned from the method.

deleteTask() is used to delete a task from salesforce. It accepts the record id of the task to be deleted as a parameter and delete that task. It returns true if the task is succesfully deleted. Otherwise, returns false.

JS Snippet (todo.js)

```
/*
*    Author: Rahul Malhotra
*    Source: SFDC Stop (https://www.sfdcstop.com)
*    Description:- Todo List Component
*    URL:- https://github.com/rahulmalhotra/lwc-sfdcstop
*/
import { LightningElement, track, wire } from 'lwc';
import getTasks from '@salesforce/apex/ToDoListController.getTasks';
import { refreshApex } from '@salesforce/apex';

export default class Todo extends LightningElement {

    // * Array to store all the todo tasks
    @track
```

```
    todoTasks = [];

    todoTasksResponse;

    // * Variable to store the new task that you want to add to the list
    newTask = '';

    /*
    *   This method is used to update new task variable
    *   with the value specified in the input field
    */
    updateNewTask(event) {
        this.newTask = event.target.value;
    }

    /*
    *   This method is used to add the value of new task variable
    *   to the list of todo tasks. It'll also clear the input field
    *   by clearing the value of newTask variable after it has been added to list
    */
    addTaskToList(event) {

        /*
        *   Unshift function - used to add element at the beginning of the array
        *   Uncommment this to use the unshift function and comment the below push functio
        */
        /*
        this.todoTasks.unshift({
            id: this.todoTasks.length + 1,
            name: this.newTask
        });
        */

        // * Push function - used to add element at the end of the array
        this.todoTasks.push({
            id: this.todoTasks.length + 1,
            name: this.newTask
        });
        this.newTask = '';
    }
```

```
    /*
    *   This method is used to delete the task from todo list
    *   based on the task id
    */
    deleteTaskFromList(event) {

        let idToDelete = event.target.name;
        let todoTasks = this.todoTasks;
        let todoTaskIndex;

        /*
        *   Method 1 - Finding the index of the task to be deleted
        *   and deleting it using the below command
        */
        for(let i=0; i<todoTasks.length; i++) {
            if(idToDelete === todoTasks[i].id) {
                todoTaskIndex = i;
            }
        }

        // * Comment the below line if you're using one of the two approaches given below
        todoTasks.splice(todoTaskIndex, 1);

        /*
        *   Un-Comment any one of the two below methods
        *   which are used to directly splice or delete
        *   the element from the array based on the index.
        *   We're finding the index by using the findIndex()
        *   function available in JavaScript
        */

        // * Method 2
        /*
        todoTasks.splice(
            todoTasks.findIndex(function(todoTask) {
                return todoTask.id === idToDelete;
            })
            , 1
        );
        */
```

```
        // * Method 3
        // todoTasks.splice(todoTasks.findIndex(todoTask => todoTask.id === idToDelete),
    }

    /*
    *   This method is used to get the list of todo tasks
    *   from apex controller when the component is initialized
    *   and update the todoTasks js array
    */
    @wire(getTasks)
    getTodoTasks(response) {
        this.todoTasksResponse = response;
        let data = response.data;
        let error = response.error;
        if(data) {
            console.log('data');
            console.log(data);
            this.todoTasks = [];
            data.forEach(task => {
                this.todoTasks.push({
                    id: this.todoTasks.length + 1,
                    name: task.Subject,
                    recordId: task.Id
                });
            });
        } else if(error) {
            console.log('error');
            console.log(error);
       }
    }

    /*
    *   This method is used to refresh the wire method response
    *   i.e. todoTasks in the browser cache
    */
    refreshTodoList() {
        /*
        *   It'll refresh the data in browser cache only
        *   if there is a change on the server side
        */
        refreshApex(this.todoTasksResponse);
```

```
    }


}
```


Let's have a look at the wire method first which is the second last method in the above code. It's calling the same **getTasks** method which is imported from the apex controller like we did in the previous tutorial, the only difference here is that, instead of binding a variable, we've binded a function with the wire method. The response object in this case (which is having **data** and **error** as properties) is not directly assigned to a variable but it's coming as a parameter named **response**in the **getTodoTasks()** method which is binded with wire service. This response is stored in the **todoTasksResponse** variable which we've defined in the beginning of the class. After that we've stored the data and error from the response object into **data** and **error** variables for simplifying code.

If we have the **data** received successfully, we're re-initializing the **todoTasks** variable to empty array and pushing all the tasks one by one by using the foreach loop on the data array. The **id** of each todo task will be the (total array length + 1) (to keep it unique), the **name** of each todo task will be the **Subject** of the task received from salesforce and the **recordId** of each todo task will be the id of the task received from Salesforce. We're storing **recordId** here as well as it'll help us to delete the todoTask from salesforce in our next tutorial.

In case of an **error** we're displaying that in the console.

We've added a **refreshTodoList()** method as well which will be used to refresh the todo list from server on clicking the refresh/sync button. This method is calling the **refreshApex()** imported from **@salesforce/apex** which is used to refresh the response of wire method stored in browser cache. We're passing **todoTasksResponse** in this method which is nothing but the response received in our wire method.

I've also reset the **addTaskToList()** from Tutorial 3 as discussed in the previous tutorial that we don't be retaining/using any code of Tutorial 4 in further tutorials.

Note:- Try to add a new task to your todo list. You'll notice that this time you'll be able to mutate/update the todo list as we haven't directly binded the todoTasks array to wire service. Instead we've received the tasks list (response) in the function as a parameter and updated the todoTasks array manually.

HTML Snippet (todo.html)

```
<!--
*    Author: Rahul Malhotra
*    Source: SFDC Stop (https://www.sfdcstop.com)
*    Description:- Todo List Component
```

```
*    URL:- https://github.com/rahulmalhotra/lwc-sfdcstop
-->
<template>
    <!-- Lightning Card -->
    <lightning-card title="ToDo List" style="width: 100%;">
        <!-- Sync Action -->
        <lightning-button-icon
            slot="actions"
            icon-name="utility:sync"
            alternative-text="Refresh"
            onclick={refreshTodoList}
            title="Refresh">
        </lightning-button-icon>
        <lightning-layout>
            <lightning-layout-item size="12" padding="around-small">
                <!-- Iterating the ToDo Tasks and displaying them in a list -->
                <ul class="slds-has-dividers_around-space">
                    <template for:each={todoTasks} for:item="todoTask">
                        <li class="slds-item" key={todoTask.id}>
                            <lightning-layout vertical-align="center" horizontal-align="s
                                <lightning-layout-item padding="horizontal-small">
                                    {todoTask.name}
                                </lightning-layout-item>
                                <lightning-layout-item padding="horizontal-small">
                                    <!-- Button to delete/remove the task from the list -
                                    <lightning-button-icon
                                        icon-name="utility:delete"
                                        alternative-text="Delete"
                                        onclick={deleteTaskFromList}
                                        name={todoTask.id}
                                        title="Delete">
                                    </lightning-button-icon>
                                </lightning-layout-item>
                            </lightning-layout>
                        </li>
                    </template>
                </ul>
            </lightning-layout-item>
        </lightning-layout>
        <p slot="footer">
            <lightning-layout pull-to-boundary="small">
```

```
            <!-- Input text field to enter the Label of new Task -->
            <lightning-layout-item padding="horizontal-small" flexibility="grow">
                <lightning-input
                    type="text"
                    placeholder="Enter new task"
                    variant="label-hidden"
                    value={newTask}
                    label="New Task"
                    onchange={updateNewTask}
                    required>
                </lightning-input>
            </lightning-layout-item>
            <!-- Button to add New Task to the ToDo List -->
            <lightning-layout-item padding="horizontal-small">
                <lightning-button
                    variant="brand"
                    label="Add"
                    onclick={addTaskToList}>
                </lightning-button>
            </lightning-layout-item>
        </lightning-layout>
        </p>
    </lightning-card>
</template>
```

This Gist brought to you by gist-it. force-app/main/default/lwc/todo/todo.html view raw

We've reset the HTML code also from Tutorial 3 and have done some important changes. If you see the refresh button (lightning-button-icon) in the beginning of **lightning:card**, you'll notice that it's has an **onclick** attribute with the value as:- **refreshTodoList** as it'll call this method from js whenever the button is clicked to refresh the todo list. I have also updated the **alternative-text** and **title** from Sync to **Refresh**.

Rest all the code is same as it was in Tutorial 3. We've done the main effort in js to link wire service with a function and we've also used refreshapex which will be called when the refresh button is clicked.

Give it a try and let me know your observations in the comments down below.

Important Observation:- If you see carefully while testing, you'll notice that the **refreshApex()** is only refreshing the todo list, when there is a change on the server side (Try inserting a new task record directly in salesforce and click on the refresh button 2-3 times in your todo list to check how many times the wire service is refreshed). Basically, the

refreshApex() only update the browser cache when there is a change on the server side otherwise it won't do anything.

So, your apex calls are preserved.