What is a Lightning Data Service?

Salesforce Lightning provides a whole list of features. One of the most common among them is **Lightning Data Service(LDS)**.

LDS is a simple and logical way to communicate with the server. With the advent of the LDS, CRUD operations have become more feasible. As noted before, normally you would have to use an apex controller to communicate with the server. However, with the LDS at your disposal, this is not necessary. You can communicate with the server without an apex class creation. In short, LDS acts as a**data layer** for Salesforce Lightning.

LDS features automatic caching and sharing of records that you can load in the LDS itself. Another functional advantage of LDS can be seen in terms of comparison of the platforms in Salesforce. It is a known fact that Salesforce has two environments: Salesforce Classic and Salesforce Lightning. While using Salesforce Classic, in order to collect customer's data to display it on the Visualforce display page, you use a standard controller. Likewise, when using Salesforce Lightning, the standard controller, in this case, is the LDS.

These are the major features of the LDS. Now let's see when and how you can use LDS for your benefit.

When can you use LDS

There are various ways by which the *Lightning Data Service* can be used. For applications in Lightning, LDS creates reads, updates or deletes (CRUD) records.

Earlier if you have to create, or, read, or update or delete a particular record, you would have had to call the server for help. A lot of time is lost and this whole operation involves page loads, callbacks, loading etc. However, with the LDS, you can avoid these. You will not be required to call the server for each operation being performed on the same data record.

Performance Benefits:

- Data access using the LDS is very simple when compared to using a server-side Apex controller.
- Highly efficient local storage acts as the foundation for LDS. Furthermore, every component using the local storage can share it.
- High reduction of complexity in operations. This is because your component consolidates every bit of your data access code.

A scenario when to use LDS

In the case of displaying and updating a particular data record in Lightning, you need to create two components. One component is for displaying the record data and the other is for updating it.
Since there are two components seeking to work on the same record data, both the components will call the server.

Due to this, there will be a **duplicate** call to the server for the same record. You can avoid this completely when using the LDS as the standard controller.

How to Use the LDS?

There are particular rules to follow when using LDS in Lightning.
First and foremost, the tag **'force: record Data'** should be used for the code that follows. This tag helps in providing record data to all your components. Furthermore, it also manages the local cache.

Example:

In the following case, you have to display and update the record data in the record data detail page.

Steps:

Create two Lightning components. One component should display the record data. Another component is for updating the record data.

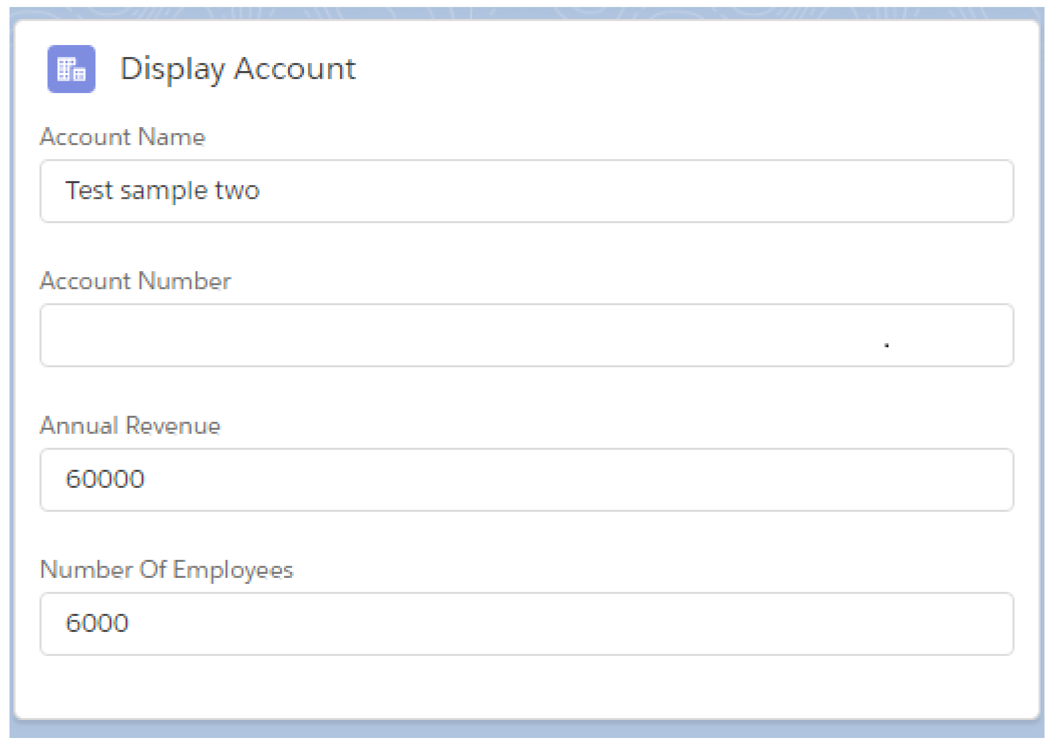
1. AccountDisplayPage.cmp

Include **'force: recordData'** in the component to make the record available. To display the record data include some lightning UI elements.

```
<force:recordData aura:id="recordEditor"
                  layoutType="FULL"
                  recordId="{!v.recordId}"
                  targetError="{!v.recordError}"
                  targetRecord="{!v.record}"
                  targetFields ="{!v.simpleRecord}"
                  mode="VIEW" />

<!-- Display a lightning card with details about the record -->
<div class="Record Details">
    <lightning:card iconName="standard:account" title="Display Account">
        <div class="slds-p-horizontal--small">
            <lightning:input label="Account Name" value="{!v.simpleRecord.Name}"/>
            <br/>
            <lightning:input label="Account Number" value="{!v.simpleRecord.AccountNumber}"/>
            <br/>
            <lightning:input label="Annual Revenue" value="{!v.simpleRecord.AnnualRevenue}"/>
            <br/>
            <lightning:input label="Number Of Employees" value="{!v.simpleRecord.NumberOfEmployees}"/>
            <br/>
        </div>
    </lightning:card>
</div>
```

Here, loading the data is also done by including **'force:recordData'** in the component.

Result of AccountDisplayPage:



Account Name: Test sample two

Account Number:

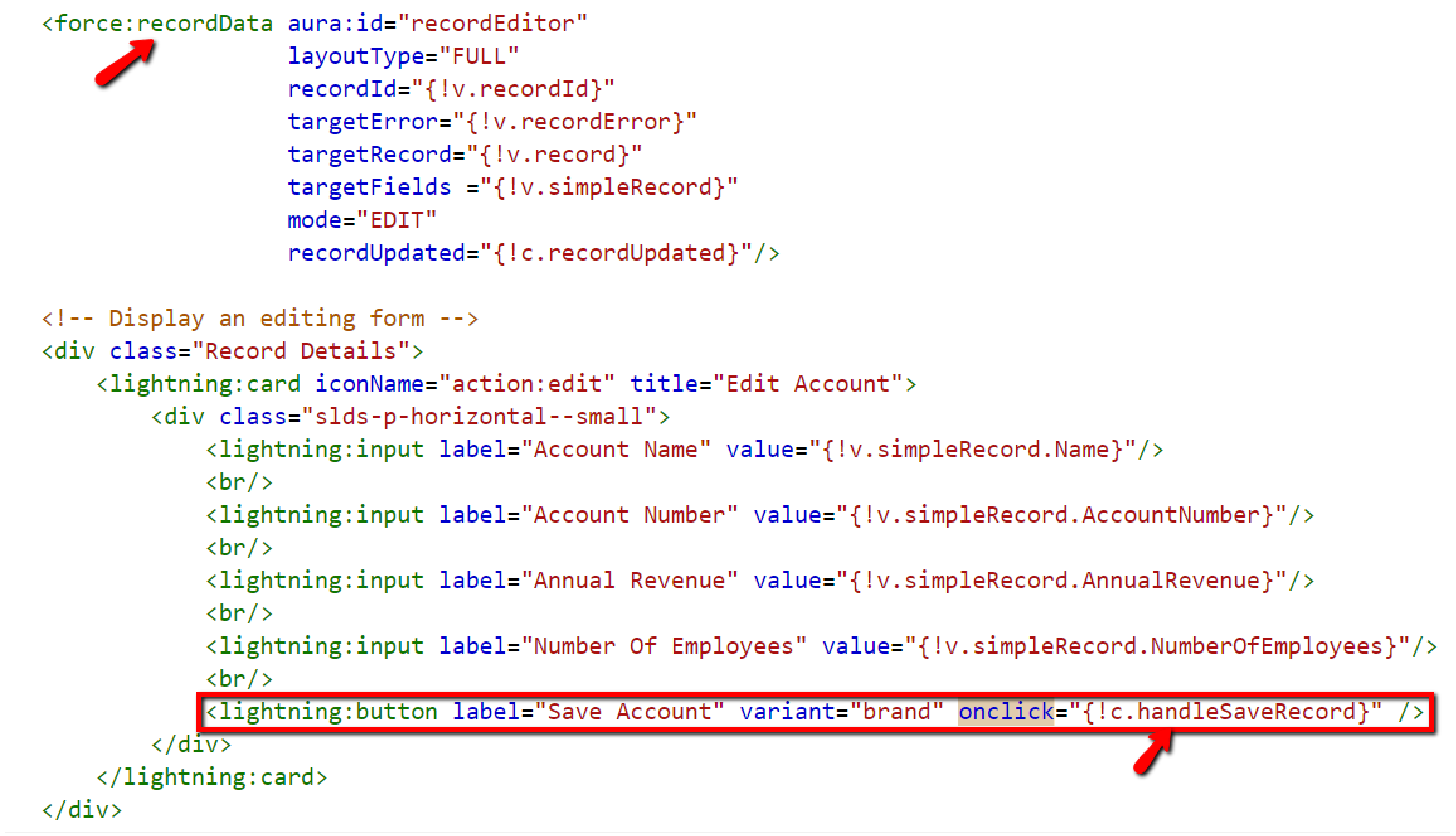
Annual Revenue: 60000

Number Of Employees: 6000

2. AccountUpdatePage.cmp

Include **'force: recordData'** and include some UI elements for update view.

Include the lightning button **'Save Account'**.



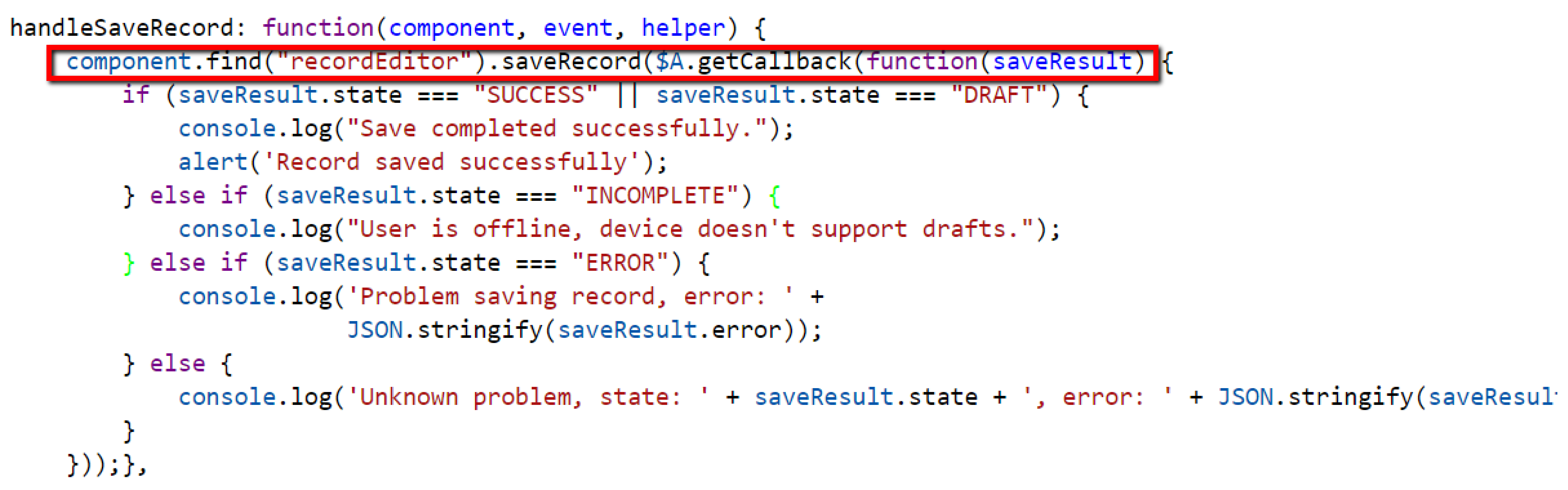
```
<force:recordData aura:id="recordEditor"
                  layoutType="FULL"
                  recordId="{!v.recordId}"
                  targetError="{!v.recordError}"
                  targetRecord="{!v.record}"
                  targetFields ="{!v.simpleRecord}"
                  mode="EDIT"
                  recordUpdated="{!c.recordUpdated}"/>

<!-- Display an editing form -->
<div class="Record Details">
    <lightning:card iconName="action:edit" title="Edit Account">
        <div class="slds-p-horizontal--small">
            <lightning:input label="Account Name" value="{!v.simpleRecord.Name}"/>
            <br/>
            <lightning:input label="Account Number" value="{!v.simpleRecord.AccountNumber}"/>
            <br/>
            <lightning:input label="Annual Revenue" value="{!v.simpleRecord.AnnualRevenue}"/>
            <br/>
            <lightning:input label="Number Of Employees" value="{!v.simpleRecord.NumberOfEmployees}"/>
            <br/>
            <lightning:button label="Save Account" variant="brand" onclick="{!c.handleSaveRecord}" />
        </div>
    </lightning:card>
</div>
```

To update the view, once more you have to add the **'force: recordData'** tag as well as a few UI elements. Also, include the lightning button **'Save Account'**.

On clicking Save Account, JavaScript controller **'handleSaveRecord'** function will be called.

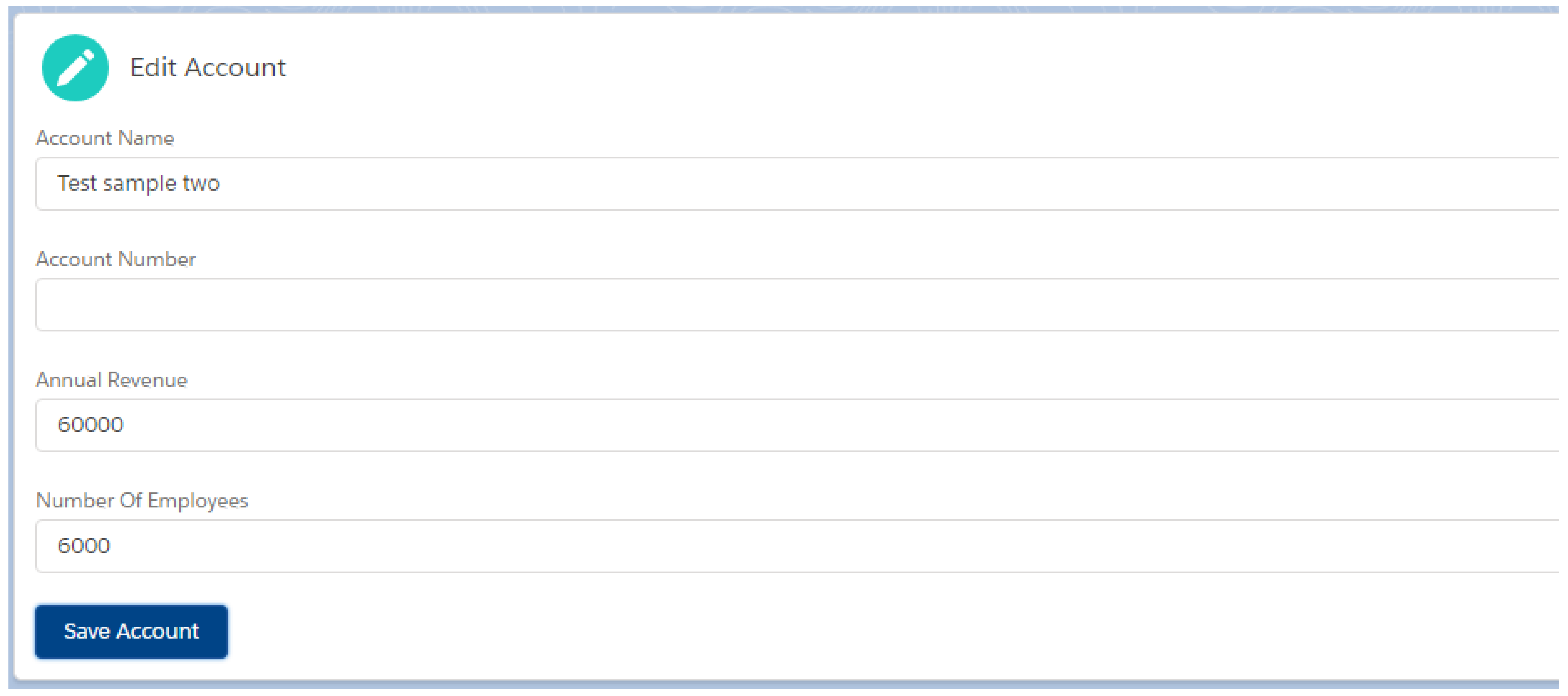
Therefore, the 'save' function is dealt with within the JavaScript controller itself. Hence, there is no longer a requirement to create the apex controller.

AccountDisplayPageController.js



```
handleSaveRecord: function(component, event, helper) {
    component.find("recordEditor").saveRecord($A.getCallback(function(saveResult) {
        if (saveResult.state === "SUCCESS" || saveResult.state === "DRAFT") {
            console.log("Save completed successfully.");
            alert('Record saved successfully');
        } else if (saveResult.state === "INCOMPLETE") {
            console.log("User is offline, device doesn't support drafts.");
        } else if (saveResult.state === "ERROR") {
            console.log('Problem saving record, error: ' +
                        JSON.stringify(saveResult.error));
        } else {
            console.log('Unknown problem, state: ' + saveResult.state + ', error: ' + JSON.stringify(saveResul
        }
    }));},
```

Result of AccountUpdatePage


Edit Account

Account Name

Test sample two

Account Number

Annual Revenue

60000

Number Of Employees

6000

The changes on ‘Update’ component should reflect on the display component.

Include “recordUpdated” parameter in **‘force: recordData’** in Update component. Consequently, when there is an update on any field in the Update Component, this parameter calls the JavaScript function.

The changes made in the ‘update’ component should reflect in the ‘display’ component as well. To do this:

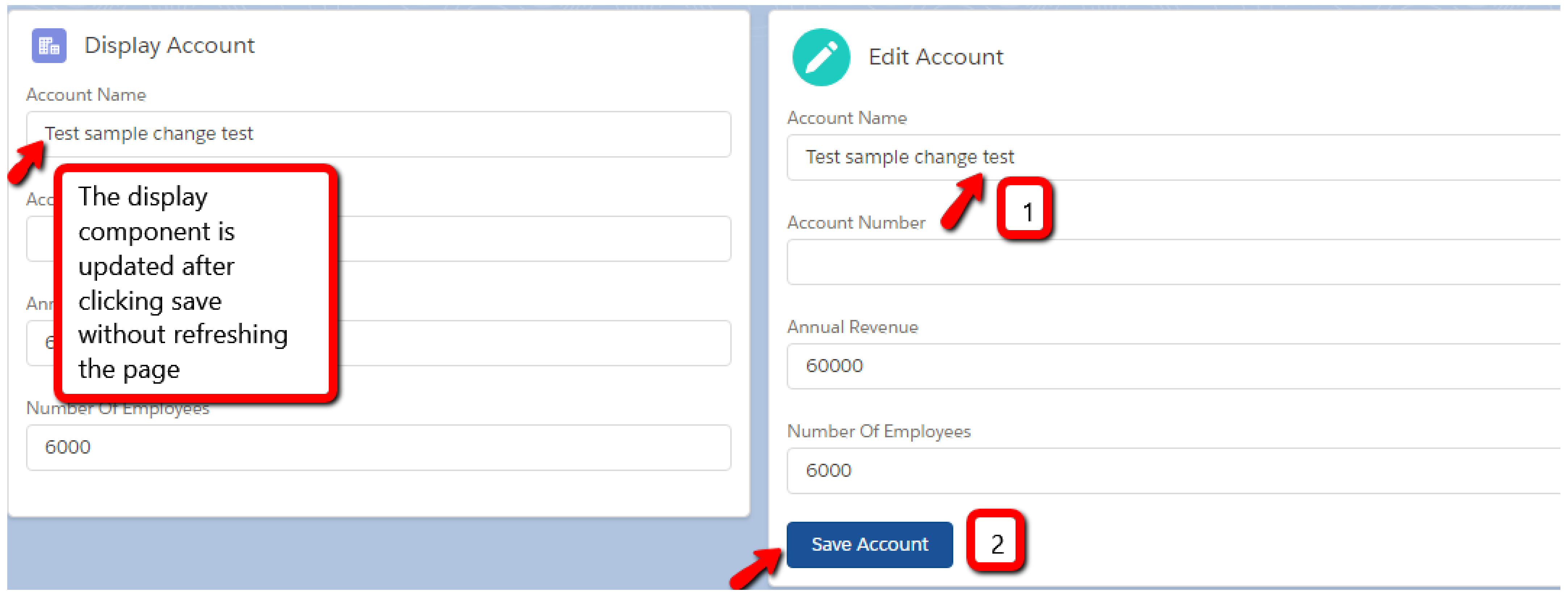
- Include ‘recordUpdated’ parameter in **‘force: recordData’** within the ‘update’ component.
- As a result, the component calls the JavaScript function when you update any field in the ‘update’ component.

```
<force:recordData aura:id="recordEditor"
                  layoutType="FULL"
                  recordId="{!v.recordId}"
                  targetError="{!v.recordError}"
                  targetRecord="{!v.record}"
                  targetFields ="{!v.simpleRecord}"
                  mode="EDIT"
                  recordUpdated="{!c.recordUpdated}"/>
```

The JavaScript function will reload the record data in the background. After this, the display component displays the update of the record data

3. AccountDisplayPageController.js

```
 recordUpdated : function(component, event, helper){
     var changeType = event.getParams().changeType;
     if (changeType === "CHANGED") {
         component.find("recordEditor").reloadRecord();
     }
 }
```


Display Account

Account Name

Test sample change test

Number Of Employees

6000

Edit Account

Account Name

Test sample change test

Account Number

Annual Revenue

60000

Number Of Employees

6000

The display component is updated after clicking save without refreshing the page
1
Save Account
2

As seen in the image above, there is an update on the changes you make as well as the display component. Both these actions take place without rendering the page.

Likewise, you can use Salesforce LDS feature to perform CRUD operations, without the use of an apex controller.