**Create a Packing List Item Component**

Create a Lightning Component to display a single item for your packing list.

* Create a component called **campingListItem**.
* Add an attribute named **item** of type **Camping\_Item\_\_c** that is required.
* Display **Name, Price, Quantity, Packed** using an expression.
* Display **Price** and **Quantity** using the appropriate formatted number fields.
* Display **Packed** status using a toggle.

**Mark Item as Packed**

Add a button to the **campingListItem** component that when clicked, marks the item as packed.

* Add a button labeled **Packed!** that calls the **packItem** controller function when clicked.
* The controller action marks the **item** attribute as packed, updates the component's item attribute using the view value provider, and disables the button.

**Create a Form to Enter New Items**

In this challenge you'll create a form to enter new items, a list to display the items entered, and add SLDS styling. First, to make our camping list look more appealing, change the campingHeader component to use lightning:layout and SLDS. Similar to the unit, style the Camping List H1 inside the slds-page-header. Add the action:goal SLDS icon using lightning:icon.

Next, modify the campingList component to contain a new item input form and an iteration of campingListItem components for displaying the items entered. Here are additional details for the modifications to the campingList component.

* Add an attribute named **items** with the type of an array of camping item custom objects.
* Add an attribute named **newItem** of type Camping\_Item\_\_c with default quantity and price values of 0.
* The component displays the Name, Quantity, Price, and Packed form fields with the appropriate input component types and values from the **newItem** attribute. The Quantity field accepts a number that's at least 1.
* Submitting the form executes the action **clickCreateItem** in the JavaScript controller.
* If the form is valid, the JavaScript controller pushes the **newItem** onto the array of existing **items**, triggers the notification that the **items** value provider has changed, and resets the **newItem** value provider with a blank sObjectType of Camping\_Item\_\_c. **For this challenge, place the code in your component's controller, not the helper.**

**Save and Load Records with a Server-Side Controller**

Persist your records to the database using a server-side controller. The **campingList** component loads existing records when it starts up and saves records to the database when the form is submitted.

* Create a **CampingListController** Apex class with a **getItems** method and **saveItem** method.
* Add a **doInit** initialization handler that loads existing records from the database when the component starts up.
* Modify the JavaScript controller to use a **createItem** method in the helper to save records to the database from a valid form submission. The new items are added to the controller's **items** value provider.

**Refactor Components and Communicate with Events**

Refactor the input form for camping list items into its own component and communicate with component events.

* Replace the HTML form in the campingList component with a new **campingListForm** component that calls the **clickCreateItem** JavaScript controller action when clicked.
* The campingList component listens for a **c:addItemEvent** event and executes the action **handleAddItem** in the JavaScript controller. The **handleAdditem** method saves the record to the database and adds the record to the **items** value provider.
* The **addItemEvent** event is of type **component** and has a Camping\_Item\_\_c type attribute named **item**.
* The campingListForm registers an **addItem** event of type **c:addItemEvent**.
* The campingListFormController JavaScript controller calls the helper's **createItem** method if the form is valid.
* The campingListFormHelper JavaScript helper creates an **addItem** event with the item to be added and then fires the event. It then resets the **newItem** value provider with a blank sObjectType of type Camping\_Item\_\_c.

--------------------------------------------------------------------------------------------------------------------------------

**Define the Expense Custom Object**

While we assume that you know how to create a custom object already, here are brief instructions, and the specifics for the Expense object and its fields.

1. Go to the Object Manager.
2. From Setup, at the top of the page, click **Object Manager**.
3. Create the custom object.
4. Select **Create** | **Custom Object**.
5. Define the Expense object.
6. Enter the following values for the object’s definition.

|  |  |
| --- | --- |
| **Field** | **Value** |
| Label | Expense |
| Plural Label | Expenses |
| Starts with vowel sound | checked |
| API Name | Expense\_\_c |

Add custom fields to the Expense object.

Scroll to the Fields & Relationships section of the object details page. For each of the following fields, click **New** and define the field with the following details.

|  |  |  |
| --- | --- | --- |
| **Field Label** | **API Name** | **Field Type** |
| Amount | Amount\_\_c | Number(16,2) |
| Client | Client\_\_c | Text(50) |
| Date | Date\_\_c | Date |
| Reimbursed | Reimbursed\_\_c | Checkbox |

As part of our challenges for this module, you'll build a camping list application. You need to do a few more tasks before passing this challenge and continuing.

* Add a custom domain with My Domain. You probably already have done this.
* Create an **Expense** custom object to work with the examples. You probably already have done this.
* Create a **Camping Item** custom object to hold the items you need to go camping with the following fields:
  1. Name (use the standard name field)
  2. Quantity: Number(18, 0) - Required
  3. Price: Currency(16, 2) - Required
  4. Packed: Checkbox(default: unchecked)