We have the catalogue rendering displaying names and images. Rendering however is only one part of the functionality we want to provide here. We also need a way to let the user pick a quantity amount, for example maybe we want tray totals for the nutritional numbers on all the items we’ve selected for an entire meal. Also, there isn’t any description currently either. We can add these capabilities to the individual item markup but we’re a little tight on screen space (especially in the mobile footprint) and it could become cluttered. One solution is to copy/expand the information to yet another page, but a better way is to just place the extra information in a **dialog**. We’ll use the built-in PrimeVue dialog component to do this:

<https://primefaces.org/primevue/showcase/#/dialog>

1. We need to update our existing item DataTable markup with the following

@row-select="onRowSelect"

Now when a user clicks on a menuitem the method **onRowSelect** will execute.

1. Then code add the actual method:

const **onRowSelect** = (event) => { try { state.selectedItem = event.data; state.dialogStatus = ""; state.itemSelected = true;

} catch (err) { console.log(err); state.status = `Error has occured: ${err.message}`;

}

};

1. We see that this method is populating 3 data variables, so we need to add those now the to the state variable. The itemSelected field is a Boolean to determine whether the dialog should be popped or not. The selectedItem is the object from the menuitems array that was clicked on, the dialogStatus is a message field that we’ll use in some upcoming code:

selectedItem: {}, dialogStatus: "", itemSelected: false,

1. Update the return statement to include the onRowSelect method

1. Add a **Dialog** component to the main.js file.

1. Add the following style to the **site.css** file:

.dialog-border { border: solid; margin: 2vw; width: 96vw;

}

1. Finally, add the dialog itself with following **code immediately after the </DataTable>**:

<Dialog v-model:visible="state.itemSelected" class="dialog-border">

<div style="text-align: center;">

<div style="margin-bottom:2vh;font-size:larger;fontweight:bold;">{{state.selectedItem.description}}</div>

<img

:src="`/burger.jpg`"

:alt="state.selectedItem.description" class="item-image"

/>

</div>

<div style="font-weight: bold; font-size: larger; margin-top: 3vh;text-align:center;">

Nutrional Info

</div>

<table style="border:solid;margin-top:2vh;">

<tr>

<td style="width:20%;font-weight:bold;">Protein</td>

<td style="width:10%;text-align:right;padding-right:3vw;">{{state.selectedItem.protein}}</td> <td style="width:20%;font-weight:bold;">Calories</td>

<td style="width:10%;text-align:right;padding-right:3vw">{{state.selectedItem.calories}}</td>

<td style="width:20%;font-weight:bold;">Carbs.</td>

<td style="width:10%;text-align:right;padding-right:3vw">{{state.selectedItem.carbs}}</td>

</tr>

<tr>

<td style="width:20%;font-weight:bold;">Fibre</td>

<td style="width:10%;text-align:right;padding-right:3vw">{{state.selectedItem.fibre}}</td>

<td style="width:20%;font-weight:bold;">Choles.</td>

<td style="width:10%;text-align:right;padding-right:3vw">{{state.selectedItem.cholesterol}}</td>

<td style="width:20%;font-weight:bold;">Salt</td>

<td style="width:10%;text-align:right;padding-right:3vw">{{state.selectedItem.salt}}</td>

</tr>

<tr>

<td style="width:20%;font-weight:bold;">Fat</td>

<td style="width:10%;text-align:right;padding-right:3vw">{{state.selectedItem.fat}}</td>

<td colspan="4">&nbsp;</td>

</tr>

</table> <div style="text-align: center" v-if="state.dialogStatus !== ''" class="dialog-status"

>

{{ state.dialogStatus }}

</div>

</Dialog>

8. Test the application now by clicking on one of the items from the list and you should see something like the following (mobile footprint):



Now that we have the details showing, we need to let the end user choose a quantity of the items they’d like to order.

This part of the application is simulating a customer going to McDonalds and ordering several food items. Since we can order different items and we can order more than one of the same item, we need to figure out the following steps (the metaphor we’re using to store everything is a tray):

* Select a quantity
* Create an “Add To Tray” button
* Store the quantity for the selected item when the button is clicked
* Confirm the selection was recorded with an add message

Let’s make those changes, to start add the quantity field now:

1. Add a PrimeVue component called **InputNumber** to main.js
2. Add a state property called qty:

qty: 0,

1. Add a style to **site.css** for the new input control:

#qty { height: 5vh !important; font-size:small; text-align: right; width: 12vw;

}

1. Add the following markup to the dialog after the existing table:

<div style="margin-top: 2vh; text-align: center">

<span style="margin-left: -10vw; margin-right: 2vw">Qty:</span>

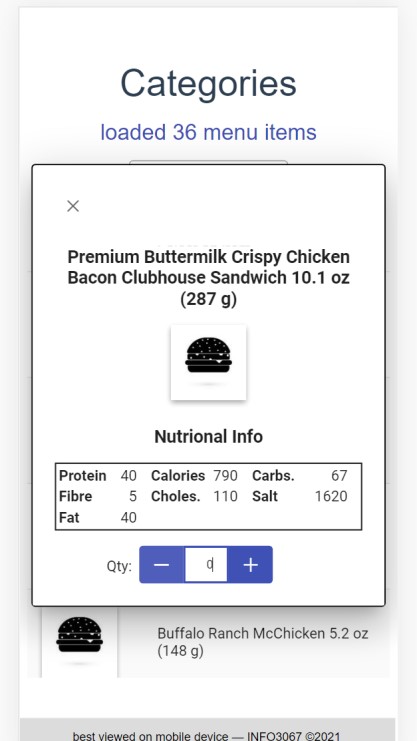
<span> <InputNumber id="qty" :min="0" v-model="state.qty" :step="1" incrementButtonClass="plus" showButtons buttonLayout="horizontal" decrementButtonIcon="pi pi-minus" incrementButtonIcon="pi pi-plus"

/>

</span>

</div>

1. Save everything and it should render like so in a mobile footprint:



1. Next let’s add the tray construct to the mix. We’ll need to be able to keep track of selections between different categories, so an easy way to do that is to set up a **sessionStorage** variable and then retrieve and return the tray information when an item is added or removed. Add the following button div to the template dialog after the qty input field (note @click is a short cut of v-on:click):

<div style="text-align: center; margin-top: 2vh">

<Button

label="Add To Tray" **@click="addToTray"**  class="p-button-outlined margin-button1"

/>

</div>

1. Add a state array variable called tray:

tray: [],

1. We’ll want to load any existing tray (there isn’t one currently) from session storage into the state variable. Update loadMenuItems method and add this code **at the end** of the method. Why we do it here will become apparent shortly.

if (sessionStorage.getItem("tray")) { state.tray = JSON.parse(sessionStorage.getItem("tray"));

}

1. Notice the button markup is calling a click event handler called **addToTray**, add this method and expose it in the return statement:

const **addToTray** = () => {

const index = state.tray.findIndex(

// is item already on the tray

(item) => item.id === state.selectedItem.id

);

if (state.qty !== 0) { index === -1 // add ? state.tray.push({ id: state.selectedItem.id, qty: state.qty, item: state.selectedItem,

})

: (state.tray[index] = {

// replace id: state.selectedItem.id, qty: state.qty, item: state.selectedItem, });

**state.dialogStatus** = `${state.qty} item(s) added`;

} else { index === -1 ? null : state.tray.splice(index, 1); // remove **state.dialogStatus** = `item(s) removed`;

} sessionStorage.setItem("tray", **JSON.stringify(state.tray)**); state.qty = 0;

};

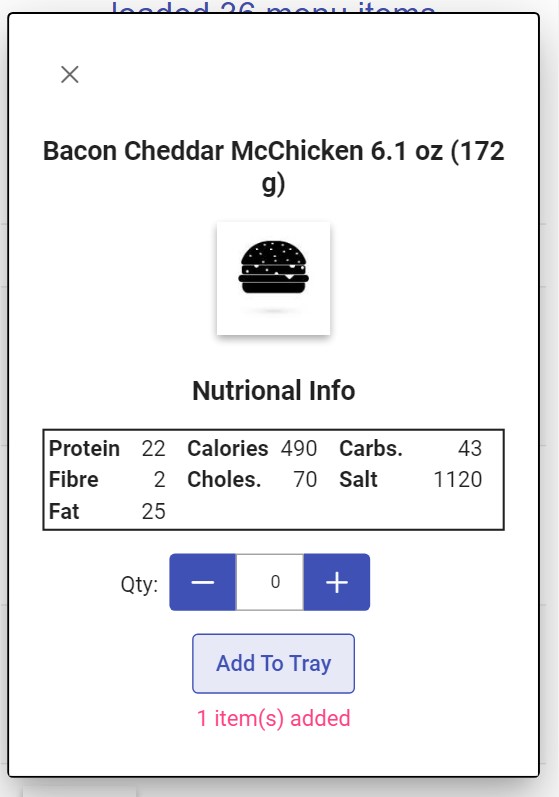
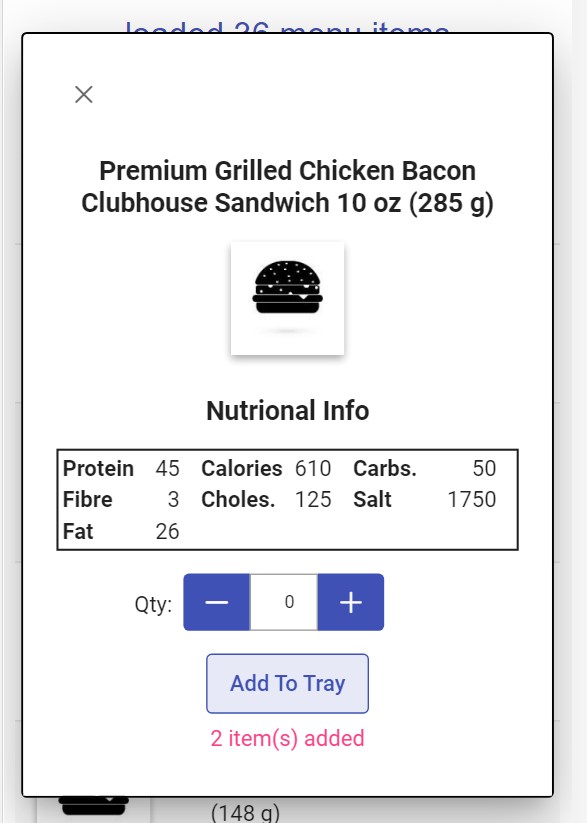
This code utilizes tray **array’s findIndex** method to determine whether or not the item already exists in the array (-1 indicates it doesn’t). Then we either add, replace or remove the item. Also, notice the tray is returned to sessionStorage as the last step.

1. This method utilizes another status field for the **dialog**. Add the **dialogStatus** variable to the **state** object and then add the following style for it:

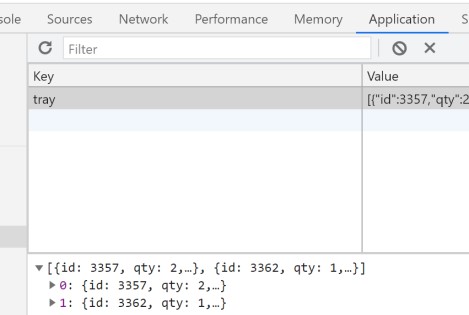
.dialog-status { font-size: 1rem; margin: 1vh; color: #ff4081; text-align:center;

}

1. Save everything and if everything works you should be able to add some items to the tray:



1. Verify the session variable has both items in it, by going to the browser’s application tab and expanding the tray variable:



# 4 – Part A

• Add 2 screenshots to a single Word document:

1. Dialog page showing **a Salad** menu item and **3 items** added (**mobile footprint**)

1. A screen shot of the Application tab command showing the contents of the session variable for tray after **adding different menu items** to it, make sure to expand out the variable before capturing the screen shot.

# 4 - Part B - Case Study Updates

Proceed now with your product catalogue so that it functions like the category/menu item setup does. Allow the user to select products by brand then display all individual products as catalogue items. Allow the user to see the details for the selected product in a dialog popup. Remember we’re not dealing with menu items and a tray in the case study, but something like a shopping cart would be the container that keeps track of products and the quantities selected.

Note, the details in the dialog for your product won’t be the same as the exercises as you should include things like **msrp**, and a lengthy **description**. Finally allow the user to select a quantity of the product for purchase.

* To format the price to currency you can create a method like so (don’t forget it to add it to the return):

const formatCurrency = (value) => { return value.toLocaleString("en-US", { style: "currency", currency: "USD",

});

};

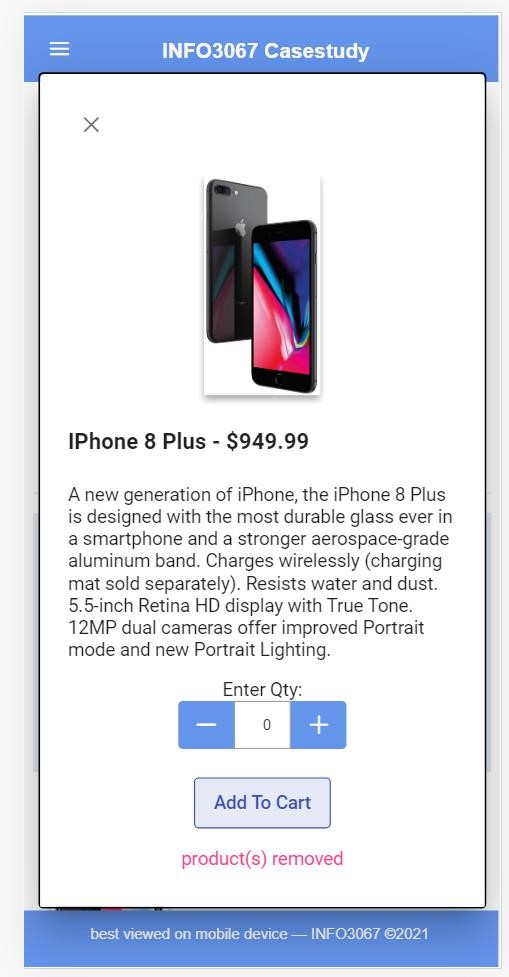
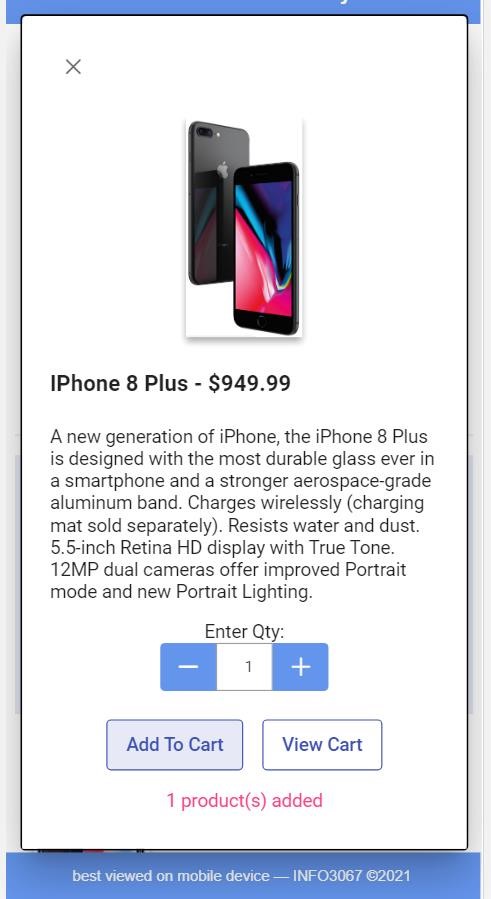
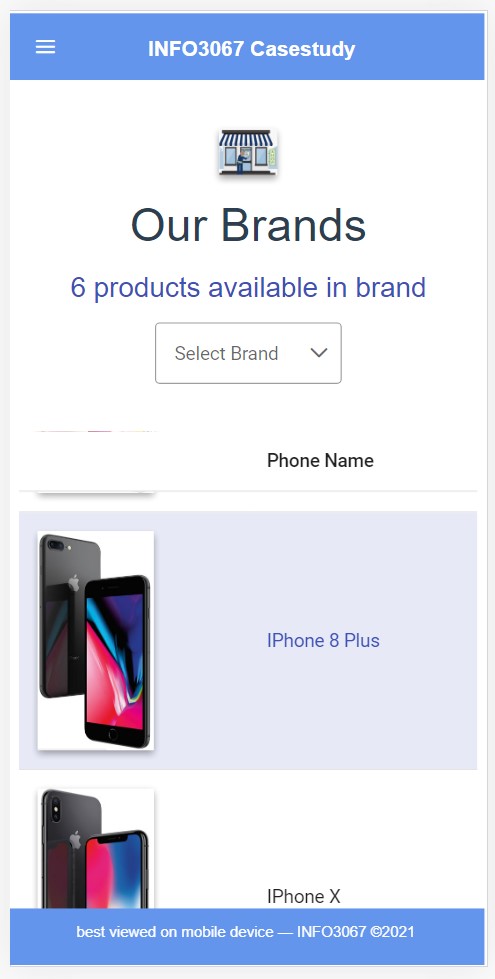
* Then call the method from the template like this:

{{ formatCurrency(state.selectedProduct.msrp) }}

An example follows for phone products (mobile footprint).

Or to remove

change the qty to 0



Add 2 more screenshots to the lab’s Word document:

1. Dialog showing **a Product** details with **2 product(s)** added (**mobile footprint**)
2. A screen shot of the application tab showing the contents of the cart after adding **3 products** to it, make sure to expand out the table values before capturing the screen shot.