Create your WEB Application with MVC paradigm and OPP – PAP support

Tutorial 20

Now, you go to introduce the Edit category. To do that, you have already the most code that you need already. You only need to do some modifications in the function get data and adapt him to edit.

Let's start. First you can use the same modal, as you use already in your create category. Only change the name to edit, and you need to send the id and edited category name in the scope of the edit button when you click to open the modal.

Follow the example from the button:

```
<td>
<!-- EDIT trigger modal -->
<button class="btn btn-primary btn-xs" onclick="openEditModal(<?php echo htmlspecialchars($category['id']); ?>,
    '<?php echo htmlspecialchars($category['category']); ?>')">
        <i class="fa fa-pencil"></i></button>
<!-- END EDIT trigger modal -->
<!-- DELETE trigger modal -->
<button class="btn btn-danger btn-xs" onclick="openDeleteModal(<?php echo htmlspecialchars($category['id'], ); ?>)">
    <i class="fa fa-trash-o "></i></button>
<!-- END DELETE trigger modal -->
</td>
```

Check the scope, you send the ID and the actual category name in the scope, of the handler function, to open the modal.

Now copy the handler open edit modal function:

```
//Function to handler the delete modal
function openEditModal(id,category) {
    // Define o valor do campo oculto no modal de exclusão
    document.getElementById('editCatId').value = id;
    document.getElementById('editCategory').value = category;
    //console.log(id);
    // Abre o modal de exclusão
    $('#editCatModal').modal('show');
}
```

Look to the scope, now, when the user click to open the edit modal, the id and the data are already inside the modal:

Now, follow the example from the edit modal:

```
<!-- Edit Modal -->
<div class="modal fade" id="editCatModal" tabindex="-1" role="dialog"
    aria-labelledby="exampleModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="exampleModalLabel">Edit category</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <form id="categoryForm">
                    <div class="form-group">
                        <label for="category-name" class="col-form-label">Category:</label>
                        <input type="text" class="form-control" id="editCategory" name="editCategory">
                        <input id="editCatId" name="editCatId" type="hidden" class="form-control" value="">
                    </div>
                </form>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-danger" data-dismiss="modal">Cancel</button>
                <button type="button" class="btn btn-primary" onclick="edit_row()">Save</button>
            </div>
        </div>
    </div>
</div>
<!-- END Edit Modal -->
```

In the first red line, you have the hidden ID, the user don't need to now tha the Id is there. And you have also the data category inside your input edit category.

Okay, next you go to create a edit row, to get the data and prepare it, and send it to the controller:

```
// Function to edit row
function edit_row()
{
    let category_input = document.querySelector("#editCategory");
    let id_input = document.querySelector("#editCatId");

    if (category_input.value.trim() === "" || !isNaN(category_input.value.trim())) {
        Swal.fire({
            icon: 'error',
            title: 'Erro',
            text: 'Por favor insira um nome de categoria válido!'
        });
        return;
    }

    let data = category_input.value.trim();
    let id = id_input.value;

    send_data({
        id: id,
        data: data,
        data_type: 'edit_category'
    });
}
```

Now, let's go to the controller. Adapt your category function and update with this code:

```
        echo json_encode($arr);
    }

    // Handle editing a category
    elseif ($data->data_type == 'edit_category') {
        $id = $data->id;
        $new_category = $data->data;

        // Call the model's edit method
        $check = $category->edit($id, $new_category);

        if ($check) {
            $arr['message'] = "Categoria editada com sucesso!";
            $arr['message_type'] = "info";
        } else {
            $arr['message'] = "Erro ao editar a categoria!";
            $arr['message_type'] = "error";
        }
        $_SESSION['error'] = "";
        $arr['data'] = "";
        $arr['data_type'] = "edit_row";

        echo json_encode($arr);
    }

    // Handle change state of a category
    elseif ($data->data_type == 'disabled_row') {
        $id = $data->id;
```

Ok, now it is time to go to the category model. Add this new code:

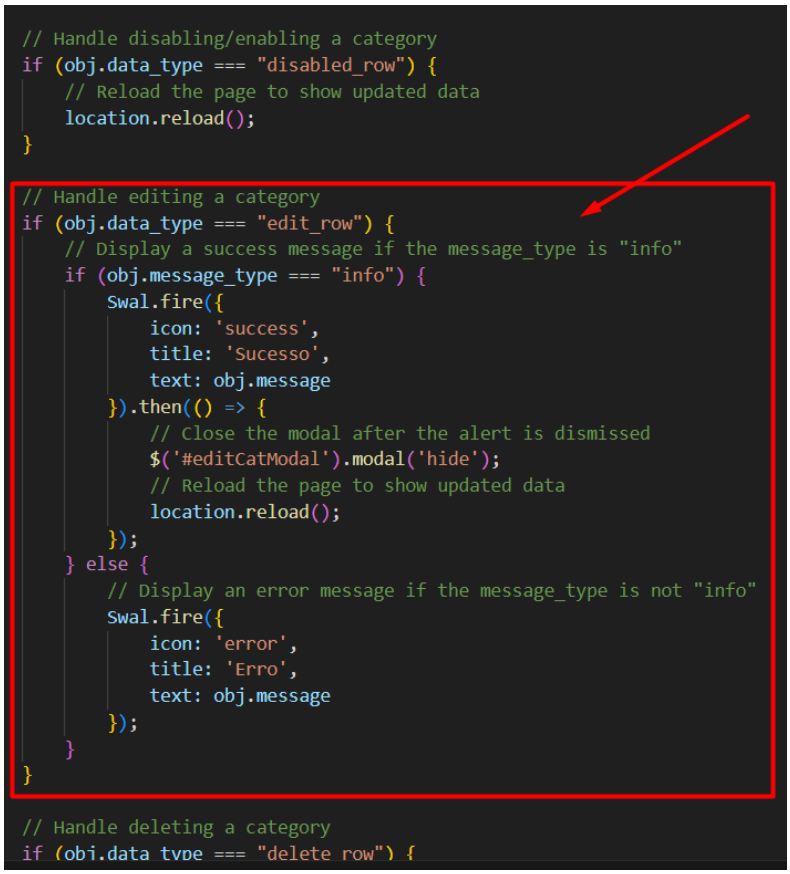
```
//Function model edit category
public function edit($id, $new_category) {
    $DB = Database::getInstance();

    // Validate the new category name
    $new_category = ucwords(trim($new_category));
    if (!preg_match("/^[a-zA-Z]+$/", $new_category)) {
        $_SESSION['error'] = "Por favor insira um nome de categoria correto!";
        return false;
    }

    // Update the category in the database
    $query = "UPDATE categories SET category = :category WHERE id = :id LIMIT 1";
    $params = array(':category' => $new_category, ':id' => $id);
    return $DB->write($query, $params);
}
```

It's quiet like the create new category, but the difference is update and not insert.

Finally, you only need to update your handler result inside your category view:



```
// Handle disabling/enabling a category
if (obj.data_type === "disabled_row") {
    // Reload the page to show updated data
    location.reload();
}

// Handle editing a category
if (obj.data_type === "edit_row") {
    // Display a success message if the message_type is "info"
    if (obj.message_type === "info") {
        Swal.fire({
            icon: 'success',
            title: 'Sucesso',
            text: obj.message
        }).then(() => {
            // Close the modal after the alert is dismissed
            $('#editCatModal').modal('hide');
            // Reload the page to show updated data
            location.reload();
        });
    } else {
        // Display an error message if the message_type is not "info"
        Swal.fire({
            icon: 'error',
            title: 'Erro',
            text: obj.message
        });
    }
}

// Handle deleting a category
if (obj.data_type === "delete_row") {
```

Check the result, and try to update one example, to validate your code.

Look, the magic is always the same:

- In the view, you get data from the input
- Prepare the data to send it to the controller
- In your controller, you receive and prepare or manipulate the data and send it to the model
- The model is responsible for making the call to the database. If it's all good, the response is okay, and the response data is returned to the controller
- Finally, the controller sends the response data to the view, and the handler shows the response to the user using SweetAlert

The flow is always the same.

M -> V-> C

Remember, always comment your code, to facilitate in the future to read them.

END Tutorial 20

Reference: **PHP Ecommerce website development | Edit category | MVC OOP - Quick programming**

Template: NICEADMIN, design by BOOTSTRAP

Modal: https://getbootstrap.com/docs/4.0/components/modal/