Word count: 999

Criterion C - Development

File handling

To setup the connection to the database, the code in the “conexion.php” file was used.

```php
<?php
$contrasena = "";
$usuario = "root";
$nombre_bd = "crud";

try {
    $bd = new PDO (
        'mysql:host=localhost;
        dbname='.$nombre_bd,
        $usuario,
        $contrasena,
        array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8")
    );
} catch (Exception $e) {
    echo "Problema con la conexion: ".$e->getMessage();
}
?>
```

I put the code in a separate file for it to work as a function, allowing cleaner reusing and separate testing. Then, for each file that used the database connection, I included:

```
include_once "Model/conexion.php";
```

Data entry and forms

Data was obtained with boostrap forms, as the one used to register an ingredient in a recipe:

```php
<form class = "p-4" autocomplete = "off" method = "POST" action = "RegisterItemToRecipe.php">
```

The autocomplete field is important, since if it was not specified, the form would save all the previous entries, even if they were incorrect, confusing the user.

The action field is filled with the name of the php file that will handle that data, sent through the POST method. There are two methods for forms: GET and POST, but GET sends the information through the URL. POST data is not saved in the browser history, and it can be all kind of data, while GET data is appended as a string in the URL and is saved into the browser history. GET reduces data privacy, and makes it possible that the client modifies the parameters passed through the URL. This is why, when modifying the database, to not to risk bugs, the POST method is used instead of the GET method.

In some cases I used GET, as when displaying the ingredients for a recipe: the ingredient name is a link to the ingredients search page, which receives through the URL the name of the ingredient type so that it displays all the items of this ingredient type in the freezer.

```
<td><a href = "SearchIngredients.php?ITYPE=<?php echo $dato->IdIngr;?>" class="
text-reset text-decoration-none"><?php echo $IType[$dato->IdIngr]; ?></a></td>
```

To access information through POST the command is $_POST[”FIELDNAME”], as in this case:

```
if(empty($_POST["RECIPE"])|| empty($_POST["oculto"])|| empty($_POST["IngrTYPE"])|| empty($_POST["IngrQTY"]) || empty($_POST["
    CRITICAL"])){
    //The empty($_POST['FIELDNAME']) returns true if that field is empty, so there is a lack of values received
    header('Location: AddIngredients.php?messageIngr=falta'); // In that case, we send an alert message through the URL
    exit();
}
```

With GET we use $_GET[”FIELDNAME”], as in ‘SearchIngredients.php’ when it receives the name of the ingredient type coming from the ingredients’ display in the recipes.

```
if(isset($_GET["ITYPE"])){
  ?>
  <div class = "display-7" align = "center" style = "color:CadetBlue"><?php echo "Items with ingredient type ".$IType[$
    _GET["ITYPE"]];?></div>
  <?php
  $Id = $_GET["ITYPE"];
```

There are several data types, so different formats are used. The ingredient and recipe names are strings, but in order to allow the user to know the allowed options, the form displays the text options compatible with the text that is being written. This is done with the field ‘list’, which is filled with all the recipe names in the recipes table, as it is visible in the following figure. The same method is used for the ingredient types, the place of the items, and to know if the ingredient is critical for a recipe. The recipe name and ingredient type are later converted to an integer: ther Id.

```
<label class = "form-label">Recipe: </label>
<input class = "form-control" name = "RECIPE" list = "recipes">
<datalist id = "recipes"> <!-- this way, we have a list of suggerences-->
<?php
    include_once "Model/conexion.php";
    $ask = $bd -> query("select * from recipes");//generating a list of recipe names
    $recipe = $ask->fetchAll(PDO::FETCH_OBJ);
    foreach($recipe as $concrete){
    //The string entered has to coincide exactly with the name of the recipe, so options
    //of the existing recipes are offered. The client still might input an incorrect name
    //but in that case the error is detected and an alert is sent, and this method facilitates
    //entering a correct name
?>
    <option value = "<?php echo $concrete->Description; ?>">
        <!--output the recipe name as an option-->
<?php
    }
?>
</datalist>
```

String forms, as the quantity of an ingredient in a recipe, the quantity of an item, and the notes of items, use `type = "text"`.

```
<div class="mb-3">
    <label class="form-label">Quantity: </label>
    <input type="text" class="form-control" name = "IngrQTY" autofocus>
</div>
```

For the consumption date of an item, we use `type = "date"`.

```
<div class="mb-3">
    <label class="form-label">Consumption date: </label>
    <input type="date" class="form-control" name = "BBDATE" autofocus>
</div>
```

To send the data, we use a submit button, which incorporates a hidden value (`type = "hidden"`).

```
<div class="d-grid">
    <input type="hidden" name="oculto" value="1">
    <input type="submit" class="btn btn-primary" value="Register">
</div>
```

1 is the default value for hidden. However, when editing data in 'EditItem.php', the file consists entirely on the editing form, and the hidden value saves the Id of the item, so that later the SQL query can access a concrete item. The same is done when editing recipes.

```
<div class="d-grid">
    <input type="hidden" name="codigo" value="<?php echo $IdItem; ?>">
    <input type="submit" class="btn btn-primary" value="Change">
</div>
```

Data addition

When adding data, variables with the values to add are obtained through POST, and then the following commands are used to insert them into the database.

```
$sentencia = $bd->prepare("INSERT INTO inventory(IdIngr, Quantity, DueDate, Place, Notes) VALUES(?, ?, ?, ?, ?);");
$resultado = $sentencia->execute([$foundId, $Quantity, $Date, $Place, $Notes]);
```

In each case, we use the instruction `$bd->prepare("INSERT INTO databaseName(fields) VALUES (? for each unknown)")` followed by `$sentencia->execute([List of variables corresponding to each field in order])`.

Data search and display

Data is displayed in tables. The class ‘table align-middle’ is used, as in this case

```
<table class="table align-middle">
  <thead> <!-- Header row with the titles -->
    <tr>
      <th scope="col">#</th>
      <th scope="col">Ingredient</th>
      <th scope="col">Quantity</th>
      <th scope="col">Preferred consumption date</th>
      <th scope="col">Place</th>
      <th scope="col">Notes</th>
      <th scope="col" colspan="2">Options</th>
    </tr>
  </thead>
  <tbody>
    <!-- Create array of ingr types-->
    <?php ...
    ?>
    <?php
      foreach($ingrediente as $dato){
    ?>
    <!--A row for each recording -->
    <tr>
      <td scope="row"><?php echo $dato->IdInv; ?></td>
      <td><?php echo $IType[$dato->IdIngr]; ?></td>
      <td><?php echo $dato->Quantity; ?></td>
      <td><?php echo date("j F Y", strtotime($dato->DueDate)); ?></td>
      <td><?php echo $dato->Place; ?></td>
      <td><?php echo $dato->Notes ?> </td>
      <td><a class="text-success" href="editItem.php?codigo=<?php echo $dato->IdInv; ?>"><i class="bi
      bi-pencil-square"></i></a></td>
      <td><a onclick="return confirm('Do you really want to delete this?');" class="text-danger" href="
      deleteItem.php?codigo=<?php echo $dato->IdInv; ?>"><i class="bi bi-trash"></i></a></td>
    </tr>
    <?php ...
    ?>
  </tbody>
</table><!-- -->
```

To extract all the records in a table, I used the command `$bd->query(“SELECT * FROM tablename”)`, as in this example.

```
$sentence = $bd->query("SELECT * FROM ingredients");
```

In some cases, I specified conditions. In that case, I used `$bd->query(“SELECT * FROM tablename WHERE condition”)`:

```
$askItem = $bd->query("SELECT * FROM inventory WHERE IdIngr = $dato->IdIngr;");
```

After this, I looped through the selected records:

```
//select the data
$ask = $bd -> query("select * from ingredients");
$type = $ask->fetchAll(PDO::FETCH_OBJ);
$IType = array();
foreach($type as $ingr){//loop through the list of recordings extracted, where $type is each item
  $IType[$ingr->IdIngr] = $ingr->Description;
}
```

I used an array in the previous example because, when accessing the records of items or recipe-ingredient relationships, ingredient types and recipe names are stored as their Id. So I loop through the existing ingredients to register in an array the ingredient name corresponding to each Id, to later on directly output the ingredient name without

repeating the loop each time: if n is the number of records and m the number of ingredient types, complexity is reduced form $O(nm)$ to $O(n + m)$.

Data deletion

I deleted records from a table by identifying them with an Id. I used the command "`DELETE FROM tablename WHERE id = ?`" and specified the Id to delete:

```
//IdInv is the Id of the item we are deleting
$instruction = $bd->prepare("DELETE FROM inventory where IdInv = ?;"); //The ? means we will specify the value later
//now the command is executed with a concrete Id and the item is deleted
$result = $instruction->execute([$IdInv]);
```

Data update

To change data, I used the file containing the form where to add the new values and the file changing the records. In the form, I accessed that concrete recording through its id:

```
$sentencia = $bd->prepare("select * from inventory where IdInv = ?;");
$sentencia->execute([$IdItem]);
$ConcreteItem = $sentencia->fetch(PDO::FETCH_OBJ);
```

And presented a form such as the one for adding data, but adding 'required value', where I put the precedent value, so that the form would be prefilled and the user would only need to modify some fields:

```
<input class = "form-control" name = "ITYPE" list = "ingredients" required value = "<?php
    echo $IType[$ConcreteItem->IdIngr]; ?>"> <!--This form field is initially filled
with the previos ingredient type -->
```

To change records, I identified their Id, and used "`UPDATE tablename SET fields = ? where Id = ?`" followed by '`$instruction->execute([concretevalues])`'.

```
//Instructions to update the values in the table inventory
$instruction = $bd->prepare("UPDATE inventory SET IdIngr = ?, Quantity = ?, DueDate = ?, Place = ?, Notes = ? where
    IdInv = ?;" );
//Execute the instruction with concrete new values for the fields of the record having such id
$result = $instruction->execute([$ItemIngr, $ItemQuantity, $ItemDueDate, $ItemPlace, $ItemNotes, $IdItem]);
```

Exception handling

I verified that all data was in the correct format. The place of items had to be the door, the shelves, or the drawers, and the Critical field for a recipe was always 'Yes' or 'No'. See both cases in the following figures.

```php
if($_POST["IPLACE"] != "Drawers" && $_POST["IPLACE"] != "Shelves" && $_POST["IPLACE"]!="Door"){
    header('Location: AddIngredients.php?mensajeItem=place');
    exit();
}
```

```php
if(($_POST["CRITICAL"] != "Yes") && ( $_POST["CRITICAL"] != "No")){
    header('Location: SearchRecipes.php?mensaje=critical');
    exit();
}
```

Also, no field of any form should be left empty:

```php
if(empty($_POST["RECIPE"])|| empty($_POST["oculto"])|| empty($_POST["IngrTYPE"])|| empty($_POST["IngrQTY"]) || empty(
    $_POST["CRITICAL"])){
    //The empty($_POST['FIELDNAME']) returns true if that field is empty, so there is a lack of values received
    header('Location: AddIngredients.php?messageIngr=falta'); // In that case, we send an alert message through the
    URL
    exit();
}
```

When registering an ingredient type or recipe name, I did also check it did not exist before, searching in the list of ingredients or recipes any recording with such name:

```php
$query = "SELECT * FROM ingredients WHERE Description = '$newIngr'"; //The SQL command that gets the ingredients
that have the same ingredient type name
$check = $bd->query($query);
$existant = $check->fetchAll(PDO::FETCH_OBJ);
if(!empty($existant)){//If it is not empty, this means that this name already exists, so the ingredient type already
exists and should not be added
    header('Location: AddIngredients.php?mensaje=repeated');
    exit();
}
```

I also checked that ingredient types or recipe names entered by the user when adding an item or an ingredient-recipe relationship were in the database by traversing the list to find any recording the same name.

```php
$ask = $bd -> query("select * from ingredients"); //get the list of all the ingredient types
$type = $ask->fetchAll(PDO::FETCH_OBJ);
$foundId = -1;
foreach($type as $ingr){//traverse the list
    if($ingr->Description == $_POST["ITYPE"]) $foundId = $ingr->IdIngr;//If we find the ingredient type, set the Id
    value
}
if($foundId == -1){//this means that we have not found the ingredient type, so it is invalid
    header('Location: AddIngredients.php?mensajeItem=type');
    exit();
}
```

Finally, after each operation a success message is sent.

```php
$instruction = $bd->prepare("UPDATE recipesingredients SET IdIngr = ?, IdRec = ?, Quantity = ?, Critical = ? where
    IdRel = ?;" );
$result = $instruction->execute([$FoundIdIngr[0]->IdIngr, $FoundIdRec[0]->IdRec, $IngrQuantity, $Critical, $IdRel]);

if ($result === TRUE) {//The operation has been performed successfully
    header('Location: SearchRecipes.php?mensaje=edited');
} else {
    header('Location: SearchRecipes.php?mensaje=error');
    exit();
}
```

Messages are sent through GET, and depending on its contents some bootstrap buttons are displayed, as in this case:

```php
<?php
    if(isset($_GET["mensajeItem"]) and $_GET["mensajeItem"] == 'type'){
        echo'<br>';
?>
<div class = "alert alert-warning alert-dismissible fade show" role = "alert">
    <strong>Error </strong> Please introduce a valid ingredient type for the new item.
    <button type = "button" class = "btn-close" data-bs-dismiss = "alert" aria-label = "close"></button>
</div>
<?php
    }
?>
```

Dates management

The enter page displays the items with near consumption date. I generate the date 20 days after today, and format it as “Y-m-d”, the same than the string format in SQL, so that I can compare the registered items:

```php
<div class = "container pt-7 border" style = "background-color:LightPink;"> <!-- Urgent ingredients-->
  <div class = "display-6" style = "color:Sienna;">Urgent</div>
  <br>
<p>Here will appear any ingredients with an expiration date 20 days from now</p>

<!-- Complete date problems -->
  <?php
    $dateToday = date_create();
  date_add($dateToday, date_interval_create_from_date_string("20 days"));//Create the date to compare, which is
  20 days from now
  $dateFormatted = date_format($dateToday, "Y-m-d"); //Format the date in a string with the same format than the
  SQL date recordings
  include_once "Model/conexion.php"; //connect with the database
  $queryItems ="select * from inventory WHERE (DueDate < '$dateFormatted')"; //get all the items near consomption
  date
  $sentencia = $bd -> query($queryItems);
  $ingrediente = $sentencia->fetchAll(PDO::FETCH_OBJ);
  if(empty($ingrediente)){ //Display a message if there are no such items
    echo "<p>Yay! There are no ingredients with near expiration date! :)</p><br>";
  }
  else{
    //output the table and loop through the items
  ?>
```