6) PDO-Shop - Produkte im Warenkorb anzeigen

a) Erstelle eine neue Seite in den Templates: **korb.php**

Kopiere das Grundgerüst aus „registrieren.php“: lösche nach dem Kopieren alles im „container“

korb.php

templates > korb.php

```php
<?php
require __DIR__.'/../includes.php';
?>

<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Pullishop</title>
    <base href="/4ckpulli_test/">
;    <link rel="stylesheet" href="assets/css/bootstrap.css">
    <link rel="stylesheet" href="assets/css/styles.css">
</head>
<body>
  <?php
  require "navbar.php";
  ?>

<br><br><br><br>

<div class="container">
</div>
</body>
</html>
```

Füge im Container noch eine Zeile mit einer Überschrift ein, mit einem
 darunter:

```php
<div class="container">
<div class="row">
  <h2>Warenkorb</h2>
</div>
<br>

</div>
</body>
</html>
```

b)Öffne die **navbar.php** und füge den Link zu dieser Seite ein, damit man dadurch auf diese neue Seite geleitet wird.

```php
            </ul>
            <form class="d-flex">
                <a class="nav-link" href="templates/korb.php">Warenkorb (<?= $korbZahl ?>)</a>
            </form>
```

c)korb2.php

Öffne die „korb2.php“ im Ordner „functions“ und lege ganz unten eine neue Funktion an, die den eben erstellten Namen hat. Darin wird die „userId“ übergeben.

```
44  function getKorbVonKunde(int $userId)
45  {}
```

Füge gleich den Zugriff auf die Datenbank ein, damit wir nicht vergessen. Sonst würde man die $dbh später nicht erkennen:

```
44  function getKorbVonKunde(int $userId):array
45  {
46      require "database.php";
```

SQL mit JOIN:

Nun folgt die SQL-Abfrage, die einen Join beinhaltet, um auf zwei Tabellen gleichzeitig zugreifen zu können.

BEACHTE:

Bei einem JOIN muss man die Elemente genau zuordnen. Da die ID in unterschiedlichen Tabellen vorkommt, so wie auch die „anzahl“ muss man ganz genau davor mit einem Punkt getrennt die genaue Tabelle dazuschreiben, wie z.B. produkte.p_id und korb.anzahl

Hier nutzt man die „korb.anzahl“ weil sie später hochgezählt wird mit einem button „add“.

Tipp:

Daher ist zumindest bei der ID es besser, diese schon beim Anlegen in der Datenbank unterscheidbarer zu machen, wie z.B. u_id und p_id zu nennen, statt zweimal „id“.

```
44  function getKorbVonKunde(int $userId):array
45  {
46      require "database.php";
47  $sql = "SELECT produkte.p_id,name,beschreibung,farbe,preis,korb.anzahl,bild
48      FROM produkte
49      JOIN korb
50      ON korb.p_id = produkte.p_id
51      WHERE korb.u_id = ". $userId;
```

```
$sql = "SELECT produkte.p_id,name,beschreibung,farbe,preis,korb.anzahl,bild
   FROM produkte
   JOIN korb
   ON korb.p_id = produkte.p_id
   WHERE korb.u_id = ". $userId;
```

Nun wird gleich in der Funktion festgelegt, dass es sich um ein Array handelt.

Gibt daher :array nach der runden Klammer ein:

```
44  function getKorbVonKunde(int $userId):array
45  {
```

- Darunter wird ab Zeile 51 mit IF geklärt, dass ein leeres Array ausgegeben wird, wenn entweder die Abfrage falsch ist oder kein Eintrag vorhanden ist.

```
49      WHERE korb.u_id = ". $userId;
50
51  $results = $dbh->query($sql);
52  if($results === false) {
53      return [];
54  }
```

$results = $dbh->query($sql);
if($results === false){
return [];

- Ansonsten wird ein Array „$korbVonKunden“ mit den gefundenen Produkten erstellt

```
54  }
55  $korbVonKunde = [];
56
```

$korbVonKunde = [];

- Danach wird durch das gesamte „results“ gearbeitet und die gefundenen Werte in die Variable gelegt

```
55  $korbVonKunde = [];
56  while($row = $results->fetch()){
57      $korbVonKunde[]=$row;
58  }
```

while($row = $results->fetch()){
$korbVonKunde[]=$row;
}

- Diese werden dann mit „return“ zur Verfügung gestellt

```
58  }
59  return $korbVonKunde;
60  }
```

return $korbVonKunde;

d)Wechsle wieder in die Seite „korb.php“

Hier sollen nun die gefundenen Produkte aufgelistet werden.
Zuerst muss man die oben erstellte Funktion einbauen – Zeile 29

```
25      <h2>Warenkorb</h2>
26  </div>
27  <br>
28  <?php
29  $korbVonKunde = getKorbVonKunde($userId);
```

Dafür füge ein „foreach“ ein.

```
28   <?php
29   $korbVonKunde = getKorbVonKunde($userId);
30   foreach($korbVonKunde as $korbVonKundeAnzeige):?>
31           <div class="row">
32
33           </div> <!-- Ende row -->
34     <?php endforeach;?>
35
36   </div>
37   </body>
38   </html>
```

```
<?php foreach($korbVonKunde as $korbVonKundeAnzeige):?>
      <div class="row">

    </div> <!-- Ende row -->
 <?php endforeach;?>
```

Das „foreach“-Statement erfolgt vor der <row> und bezieht sich auf die Variable mit dem Namen „$korbVonKunden“. Diese wird etwas anderslautende Variable nach dem „as“ verwendet. (Dies benötigt man dann in der „korbEinzel.php“).

Das Ende der „foreach“ wird ebenfalls angeschrieben.

Dazwischen wird eine Klasse <row> eingefügt, damit nach jedem gefundenen Produkt eine neue Zeile angefangen wird.

e) Produkte anzeigen lassen

Das Layout soll die einzelnen Produkte, die ein User in den Warenkorb gelegt hat mit Bild und Text untereinander angezeigt.

Diese bekommt ein einfaches Layout mit Bootstrap-DIV. So ähnlich wie die „card.php“, die ja auch alle Produkte anzeigt.

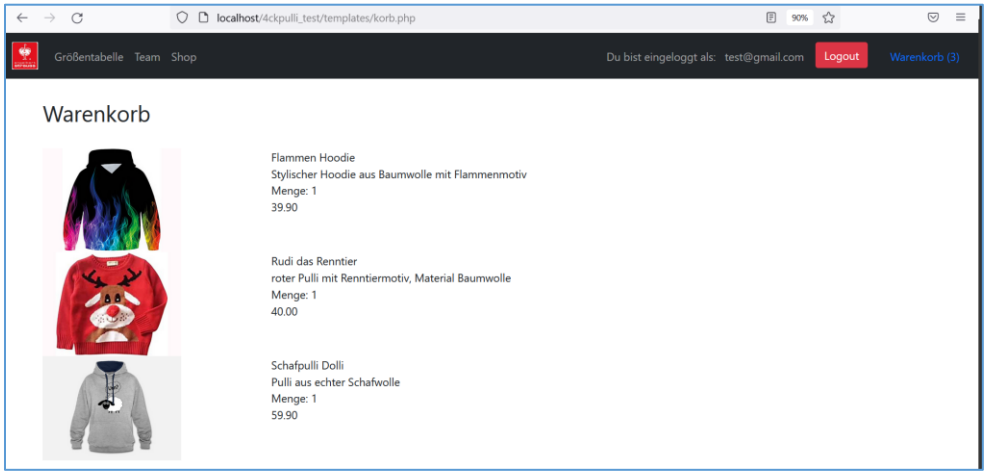
- Dabei soll das Bild links und die Beschreibung rechts danebenstehen.
- Für das Bild kopiere den Code aus der Zeile 3 aus „card.php“ und ändere
    - die col – damit sie nebeneinander sind und
    - nur den Namen der Variablen von $row auf $korbVonKundeAnzeige:

```
30   foreach($korbVonKunde as $korbVonKundeAnzeige):?>
31     <div class="row">
32
33       <div class="col-3">
34         <?php echo '<img height="150" width="200" src="'.base64_encode( $korbVonKundeAnzeige['bild'] ).'"/>'; ?>
35       </div>
36       <div class="col-6">
37         <div><?= $korbVonKundeAnzeige['name']?></div>
38         <div><?= $korbVonKundeAnzeige['beschreibung'] ?></div>
39         <div>Menge: <?= $korbVonKundeAnzeige['anzahl'] ?></div>
40         <div><?= $korbVonKundeAnzeige['preis'] ?></div>
41       </div>
42
43     </div> <!-- Ende row -->
44    <?php endforeach;?>
45
46   </div>
47   </body>
```

```
<div class="col-3">
  <?php echo '<img height="150" width="200" src="'.base64_encode(
$korbVonKundeAnzeige['bild'] ).'"/>'; ?>
</div>

<div class="col-6">
   <div><?= $korbVonKundeAnzeige['name']?></div>
   <div><?= $korbVonKundeAnzeige['beschreibung'] ?></div>
   <div>Menge: <?= $korbVonKundeAnzeige['anzahl'] ?></div>
   <div><?= $korbVonKundeAnzeige['preis'] ?></div>
</div>
```

Ergebnis:



4)Seite verschönern

Öffne die „korb.php“.

4a)Ergänze in der Zeile mit „row“ eine zusätzliche Klasse mit einem Namen wie z.B. „warenSeite“.

```
30   foreach($korbVonKunde as $korbVonKundeAnzeige):?>
31     <div class="row warenSeite">
```

Danach muss in der bereits vorhandenen CSS-Datei „style.css“ diese Klasse erstellt werden.
Es soll unter jedem Eintrag unterhalb eine Linie gezogen werden

```
24   .warenSeite {
25       border-bottom: 1px solid #DDD;
26
27   }
```

Abstand oben und unten:

```
24  .warenSeite {
25      border-bottom: 1px solid #DDD;
26      padding-top: 15px;
27      padding-bottom: 15px;
28  }
```

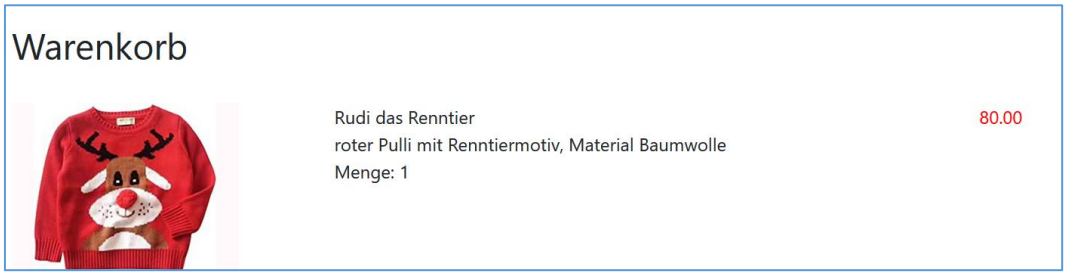
Ergebnis ist nun schon schöner:

Die Bootstrap-Grid Verteilung soll sich noch zu Gunsten einer eignen Spalte für den Preis ändern, damit dieser extra steht und besser gesehen wird:

- erstelle ein neue Spalte mit „col-2“,
- schneide die Zeile mit dem Preis aus, um sie hier einzufügen und
- reduziere die mittlere Spalte von 8 auf 6
- Damit der Preis rechtsbündig angezeigt wird, verwende die Bootstrap-Klasse „text-end“. Früher hieß es „right“, aber seit der Version 5 wird „links bzw. rechts“ umgesetzt mit „start bzw. end“
- der Preis soll eine rote Farbe bekommen, das reicht mit einem einfachen CSS-Style direkt dort in der Zeile

```
39          <div>Menge: <?= $korbVonKundeAnzeige['anzahl'] ?></div>
40        </div>
41        <div class="col-2 text-end">
42          <div style="color:red;"><?= $korbVonKundeAnzeige['preis'] ?> </div>
43        </div>
```

Ergebnis:



- Ein Eurozeichen am Ende

```
41        <div class="col-2 text-end">
42          <div style="color:red;"><?= $korbVonKundeAnzeige['preis'] ?> € </div>
43        </div>
```

https://www.youtube.com/watch?v=qOt-k7o7hS0&list=PLz858EFEcxiElOUr7b3Pql1CMHuVRYkTh&index=8 7:30

4b)Unterhalb ein Button mit „Zur Kasse gehen“

Nach dem Ende der Schleife, aber noch innerhalb der erstelle einen neue <row> und einen Link mit Button-Style. Damit ein kleiner Abstand zum Warenkorb darüber vorhanden ist, sollte eine
 eingefügt werden.

Wenn man den Button klickt, soll man auf die noch später zu erstellende Seite „bezahlung.php“ geleitet werden:

```
46    <?php endforeach;?>
47    <!-- Button zur Kassa -->
48    <br>
49    <div class="row">
50      <a href="templates/bezahlen.php" class="btn btn-primary col-12">Zur Kassa gehen</a>
51    </div>
```

Zur Kassa gehen

Knapp darüber soll die Gesamtsumme der Waren im Warenkorb angezeigt werden.

Dafür wird wieder eine <row> erstellt mit 12 Spalten und dem Text rechtsbündig:

```
46    <?php endforeach;?>
47    <!-- Button zur Kassa -->
48    <br>
49    <div class="row">
50      <div class="col-12 text-end">
51      </div>
52    </div>
```

4c)Summe Artikelanzahl

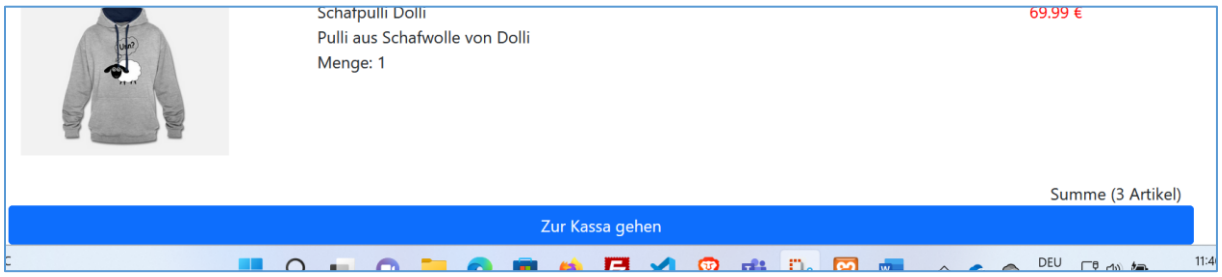
Für die Anzahl der Artikel kann man auf eine alte Variable zurückgreifen, die schon in der Datei „korb2.php“, nämlich

```
16    $korbZahl = countProdukte($userId);
```

Daher schreibe:

```
49    <div class="row">
50      <div class="col-12 text-end">
51        Summe (<?= $korbZahl ?> Artikel)
52      </div>
53    </div>
```

Ergebnis:



4d)Summe Gesamtpreis

Die Summe des Gesamtpreises soll in der Datei „korb.php“ mit einer neu zu erstellenden Variablen erstellt werden, die man sich dann holen und hier einbauen kann.

Öffne „korb2.php“.
Erstelle ganz unten eine neue Funktion, nämlich

```
62    return $korbVonKunde;
63    }
64
65    function getSummeFuerUserId(){
66
67    }
```

Dabei muss man die „userId“ übergeben und man erwartet eine Zahl als Ergebnis. In der Funktion wird man auf die Datenbank zugreifen müssen und benötigt daher eine „sql“-Abfrage. Die zuerst eine Verbindung zur Datenbank aufweisen muss:

```
62    return $korbVonKunde;
63    }
64
65    function getSummeFuerUserId(float $userId):float {
66        require "database.php";
67        $sql = "";
68    }
```

In SQL muss man die beriets vorhandene Funktion SUM() nutzen.

Einfacherweise kann man das letzte SQL-Statement **kopieren** und statt den einzelnen Elementen die Summe dieser errechnen lassen:

```
45    function getKorbVonKunde(int $userId):array
46    {
47        require "database.php";
48    $sql = "SELECT produkte.p_id,name,beschreibung,farbe,preis,korb.anzahl,bild
49        FROM produkte
50        JOIN korb
51        ON korb.p_id = produkte.p_id
52        WHERE korb.u_id = ". $userId;
```

Bearbeite nun die Zeile wo „SELECT“ steht:

```php
function getSummeFuerUserId(float $userId):float {
    require "database.php";
    $sql = "SELECT SUM(preis)
    FROM produkte
    JOIN korb
    ON korb.p_id = produkte.p_id
    WHERE korb.u_id = ". $userId;
```

Darunter wird wieder die Variable $results gesetzt. Das kann man von oben kopieren.

$results = $dbh->query($sql);

Darunter wieder eine **IF-Verzweigung**,

- nämlich, wenn das Resultat falsch ist, z.B. keine Ausgabe aus der Datenbankabfrage soll eine Null zurückgegeben werden.

```php
    $results = $dbh->query($sql);
    if($results === false) {
    return 0;
    }
}
```

- **Ansonsten** soll eine Ausgabe erfolgen. Hier genügt die einfache Form mit „fetchColumn“:

```php
    }
    return (float)$results->fetchColumn();
}
```

return (float)$result->fetchColumn();

Info: Das Problem, dass man den Wert „Null“ zurückbekommen würde, wenn keine Daten eingetragen sind, löst man mit der Konvertierung in einem Datentyp „float“. Nun wir eine Zahl zurückgegeben, die auch Kommawerte anzeigen kann..

4d)Summe des Eurobetrages ausgeben

Unterhalb der gerade erstellten Funktion (nach der schließenden Klammer) erstelle eine **neuen Variable**, die diese neue Funktion aufruft. Damit kann sie später ausgegeben werden, indem sie aufgerufen wird.

$korbSumme = getSummeFuerUserId($userId);

```php
    return (float)$results->fetchColumn();
}
// neue Variable für die Summe
$korbSumme = getSummeFuerUserId($userId);
```

Dieser Wert wird nun in die Seite des Warenkorbes übertragen, daher wechsle in die „korb.php“ um diese Variable einzutragen, die nun die Summe beinhaltet. Schreibe dahinter einfach das Eurozeichen. Man verwendet hier wieder das schon bekannte „echo“ in Kurzschreibweise:

<?= $korbSumme ?>

```
50      <div class="col-12 text-end">
51          Summe (<?= $korbZahl ?> Artikel): <?= $korbSumme ?> €
52      </div>
```

Nach einem Speichern und Aktualisieren der Warenkorbseite sollte man nun die Summe des Einkaufes ablesen können:

Ergebnis:

5)Menge der einzelnen Produkte erhöhen

Nun soll diese Anzahl erhöht werden, wenn man ein weiteres Stück dieses Produkts addiert. Nebenbei muss sich auch die Summe in Euro erhöhen, wenn ein Produkt dazukommt.

Wie:

- Übergabe der richtigen „Produkte-ID“ durch URL Verlängerung die hier „addiereZumWarenkorb“ lautet.
- Ziel der URL ist wiederum die „korb2.php“, wo auch später die Funktion dafür entsteht, die die Übergabe verarbeitet.
- Auslesen der URL
- eine neue Funktion, die um eins erhöht

5.1)Anzeige mit Button

Dazu ertelle in der „korb.php“ in der gleichen Zeile wie die Menge zwei Buttons:

```
39      <div>Menge: <?= $korbVonKundeAnzeige['anzahl'] ?></div>
40      <!-- zwei Buttons -->
41      <a href="functions/korb2.php?addiereZumWarenkorb=<?= $korbVonKundeAnzeige['p_id']?>"
42      class="btn btn-success col-3">+ 1</a>
43      <a class="btn btn-warning col-3">- 1</a>
44
45    </div>
```

<!-- zwei Buttons -->
<a href="functions/korb2.php?addiereZumWarenkorb=<?= $korbVonKundeAnzeige['p_id']?>"
class="btn btn-success col-3">+ 1
- 1

Farbe unterschiedlich, einmal „success“ und dann „warning“.

Der PHP-Teil mit der ID dient als URL-Verlängerung, die genauso wie bei „in den Warenkorb“ auf der Seite „card.php“-Zeile 22, dazu dient, die richtige Auswahl des Produktes weiterzugeben.

Diese ID wird nämlich benötigt, um das richtige Produkt zu erhöhen, und zwar in der neu zu erstellenden Funktion in der „korb2.php“.

5.2)Neue Fuktion anlegen

Öffne „korb2.php“.

Da die Übergabe der neuen Weiterleitung mittels „GET“ so ähnlich ist wie die bereits verwendete in den Zeilen 19 bis 26 machen wir uns es leichter. Kopiere diese Zeilen und füge sie ganz unten ein und verändere sie dann.

- Der Name der Weiterleitung ist nun „addiereZumWarenkorb“
- Daher isrt auch die GET genauso für die Variable §proiductId – die ja hier übergeben wird
- Den Namen der Funktion in „addEinsZumKorb“ und
- Die Weiterleitung mittels „header“ , sollte zurück zur gerade geöffneten Seite führen, da durch die Erhöhung um ein Stück die Menge ja neu ist und man ja vielleicht auch andere Produkte erhöhen will.

```
80  $korbSumme = getSummeFuerUserId($userId);
81
82  // Erhöhung der Anzahl über den Button
83  if (isset($_GET['addiereZumWarenkorb'])) {
84
85      $productId = $_GET['addiereZumWarenkorb'];
86      addEinsZumKorb($userId, $productId);
87
88      header("Location: /4ckpulli_test/templates/korb.php");
89      exit();
90  }
```

5.3)Neue Funktion erstellen

Arbeite weiter in der „korb2.php“.

Da die Funktion „addProductToKorb“ als Grundlage dienen soll, kopiere diese und füge sie gleich darunter ein.

- Ändere den Namen der Funktion auf „addEinsZumKorb“
- Alles andere kann man lassen, nur muss man eine neue Zeile einfügen nämlich: Prinzipiell muss nur die Erhöhung um +1 eingebaut werden. Sonst ist alles gleich, außer natürlich der Name, der sich ändert in „addEinsZumKorb“
- Die Erhöhung erfolgt nur durch folgenden einfachen Code:

```
ON DUPLICATE KEY UPDATE anzahl = anzahl +1";
```

ON DUPLICATE KEY UPDATE anzahl = anzahl +1

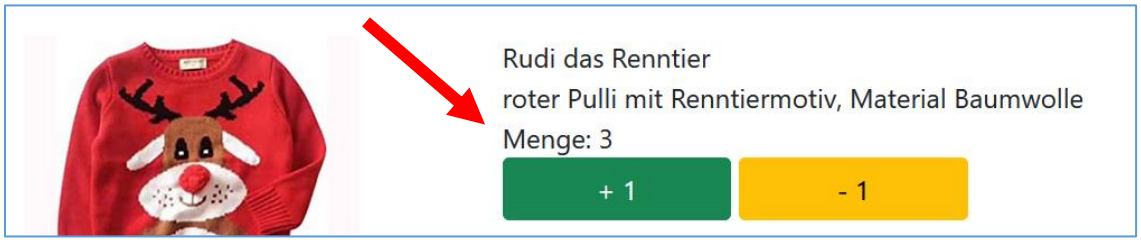
```php
// zusätzliches Warenkorb hinzufügen
function addEinsZumKorb($userId, $productId)
{
    require "database.php";
    $sql = "INSERT INTO korb SET
            p_id = :productId,
            u_id = :userId,
            anzahl=1
            ON DUPLICATE KEY UPDATE anzahl = anzahl +1";

    $statement = $dbh->prepare($sql);

    $statement->execute([
     ':productId'=> $productId,
     ':userId'=> $userId
    ]);
}
```

Testen:

Klickt man auf den gelben Button um ein Stück zu addieren, sollte sich die Zahl daneben sofort erhöhen:



auch in der Datenbank erfolgt natürlich die Erhöhung – siehe bei Anzahl: hier schon 3 Stück

	id	p_id	u_id	anzahl	created
Löschen	383	4	1	3	2022-11-05 12:48:29

https://www.youtube.com/watch?v=qOt-k7o7hS0 14:10

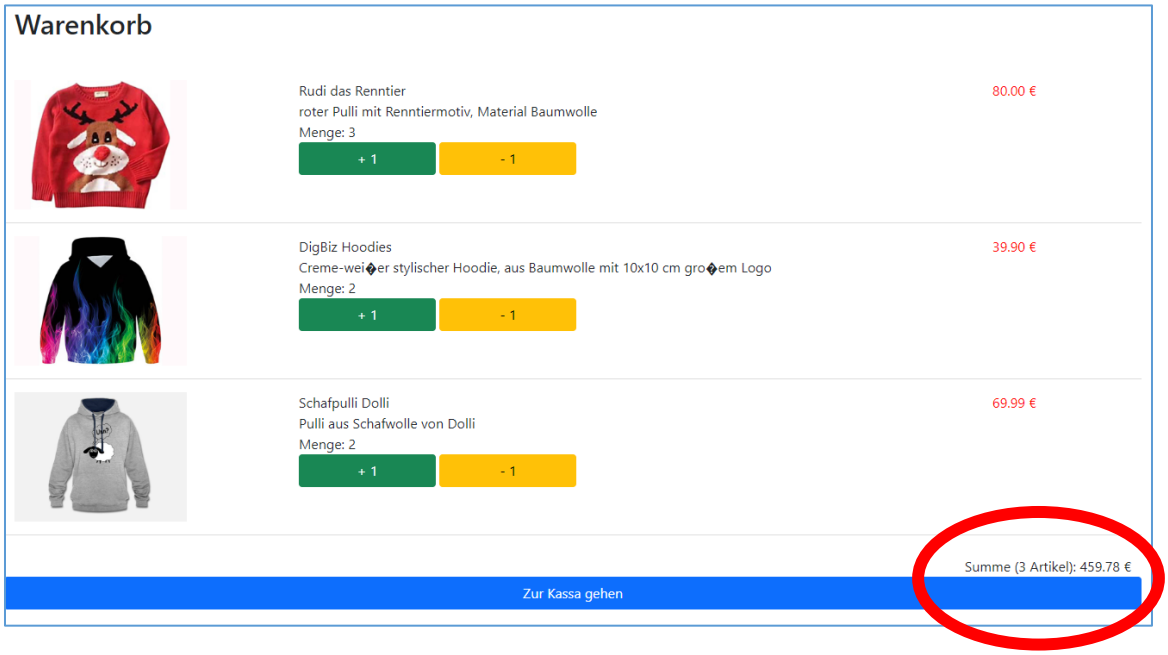
6)Summe ebenfalls erhöhen

Wenn sich die Menge erhöht, muss sich natürlich auch die Summe erhöhen.

Dafür muss man nur in der Datei **„korb2.php“** bei der „SUM“ die richtige Anzahl, also die im Korb, mit dem Preis multiplizieren:

```
65  function getSummeFuerUserId(float $userId):float {
66      require "database.php";
67      $sql = "SELECT SUM(preis * anzahl)
68      FROM produkte
69      JOIN korb
```

Die Gesamtsumme ganz unten, wo alle Produkte als Endpreis zusammengezählt werden, hat sich nun angepasst.

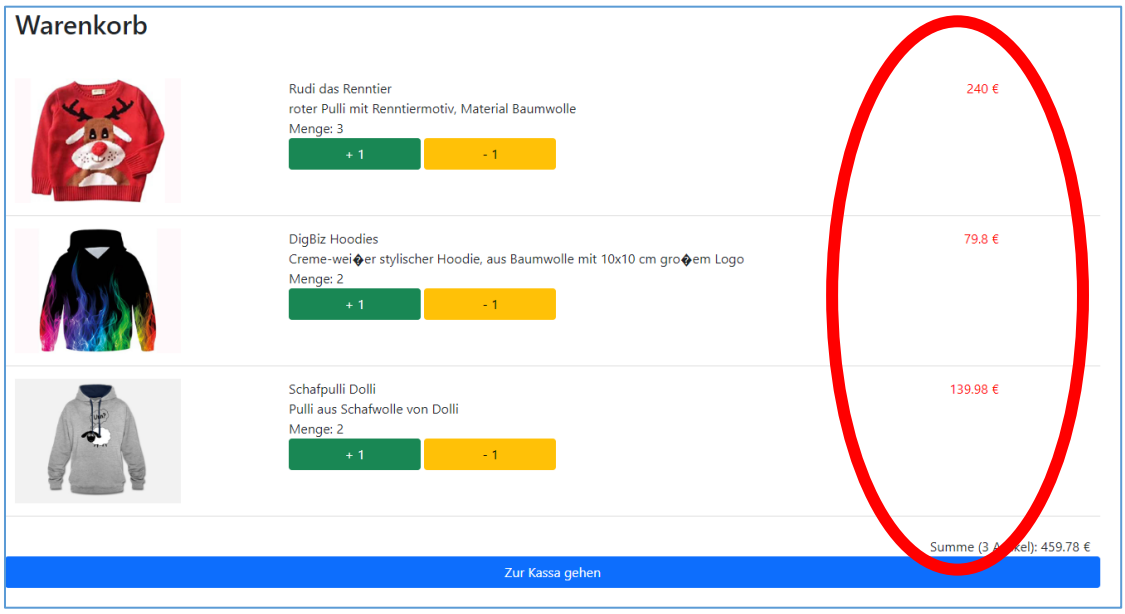


korb.php – Preis pro Produkt ebenfalls multiplizieren:

Damit sich auch die Preise der einzelnen Produktzeilen ändert, wenn mehr als eines bestellt wird, muss man noch in der Datei „korbEinzel.php“ die Anzeige des Preises mit der Menge multiplizieren:

```
46      <div class="col-2 text-end">
47          <div style="color:red;"><?= $korbVonKundeAnzeige['preis'] *
48      $korbVonKundeAnzeige['anzahl']?> € </div>
49      </div>
```

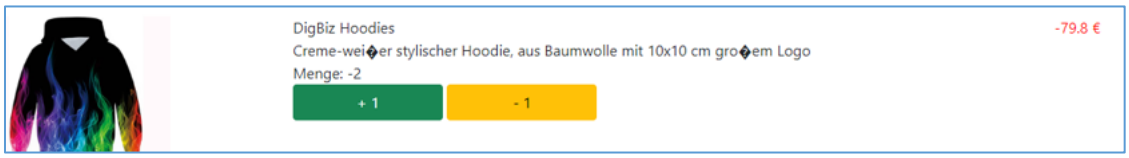
Ergebnis:



7)Menge reduzieren

Hier das gleiche wie in Punkt 5, nur mit anderen Namen und die Subtraktion statt Addition in der Funktion „subtrahiereVomWarenkorb“.

ABER: Wert NICHT UNTER NULL – sonst ja „Gutschrift“ 😊



Man muss beachten, dass der Wert nicht unter NULL kommen kann. Dafür

- Haben wir schon früher in der Datenbank vorgesorgt und in der Tabelle „korb“ den Eintrag „anzahl“ mit dem Attribut „unsigned“ versehen:

- Zusaätzlich muss man nun hier im Code noch eine IF einarbeiten, die verhindert, dass man unter NULL kommt.

Arbeite im ordner „templates“ in der „korb.php“:

```
39      <div>Menge: <?= $korbVonKundeAnzeige['anzahl'] ?></div>
40      <!-- zwei Buttons -->
41      <a href="functions/korb2.php?addiereZumWarenkorb=<?= $korbVonKundeAnzeige['p_id']?>"
42      class="btn btn-success col-3">+ 1</a>
43      <a href="functions/korb2.php?subtrahiereVomWarenkorb=<?= $korbVonKundeAnzeige['p_id']?>"
44      class="btn btn-warning col-3">- 1</a>
```

„korb2.php“: Kopiere den Bereich von oben „addiereZumWarenkorb“, füge alles darunter ein und änderen nur auf „subtrahiereVomWarenkorb“:

Wechsle in die „korb2.php“:

- Kopiere die Übergabe der eben erstellen Erhöhung und die Funktion von oben „addiereEinsZumKorb“, füge alles darunter ein und änderen nur auf „subtrahiereVomWarenkorb“ und
- ändere den Funktionsnamen auf „abziehenVomKorb“ in der „IF“ sowie im echten Funktionsnamen, der darunter erstellt wird.

```
107   }
108
109   // Abziehen von einer Produkt über den Button
110   if (isset($_GET['subtrahiereVomWarenkorb'])) {
111
112       $productId = $_GET['subtrahiereVomWarenkorb'];
113       abziehenVomKorb($userId, $productId);
114
115       header("Location: /4ckpulli_test/templates/korb.php");
116       exit();
117   }
```

Darunter ist auch die Kopie zu ändern, nämlich der Funktionsname auf „abziehenVomKorb“ und im SQL-Befehl ein Minus statt dem Plus vor dem 1er.

```
117   }
118   // Reduktion eines Produktes im Warenkorb
119   function abziehenVomKorb($userId, $productId)
120   {
121       require "database.php";
122       $sql = "INSERT INTO korb SET
123               p_id = :productId,
124               u_id = :userId,
125               anzahl=1
126               ON DUPLICATE KEY UPDATE anzahl = anzahl -1";
127
128       $statement = $dbh->prepare($sql);
129
130       $statement->execute([
131        ':productId'=> $productId,
132        ':userId'=> $userId
133       ]);
134   }
```

Nun muss noch eine IF-Abfrage verhindern, dass man ins Minus kommt bzw. dass eine Fehlermeldung erscheint.

- Unterhalb von require "database.php"; füge eine eigene SQL1 ein, die die Anzahl aus der Datenbank holt, welche zum Produkt und zum User passt.

```php
require "database.php";
$sql1 ="SELECT anzahl FROM korb WHERE
u_id =" .$userId." AND
p_id =" .$productId;
$row = $dbh->query($sql1);
$anzahl = $row->fetchColumn();
```

$sql1 ="SELECT anzahl FROM korb WHERE
u_id =" .$userId." AND
p_id =" .$productId;
$row = $dbh->query($sql1);
$anzahl = $row->fetchColumn();

- Danach hat man ja diese Anzahl und kann in der IF festlegen, dass nur bei einer positiven Anzahl der Abzug stattfinden darf.

```php
if($anzahl >= 1){

$sql = "INSERT INTO korb SET
```

- In der IF ist keine Änderung nötig, daher wird der Code so belassen.
- Erst am Ende wird die „else“ angehängt:

```php
$statement->execute([
':productId'=> $productId,
':userId'=> $userId
]);
}else {
$anzahl = 0;
}
}
```

- Die ELSE-Alternative ist, dass die Anzahl einem Wert von Null entspricht und keine weitere Reduktion somit möglich ist.

Testen: