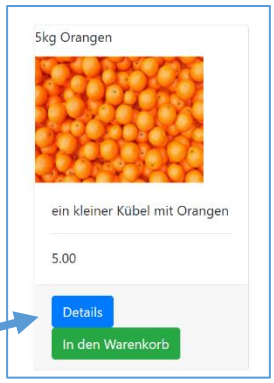
Detailseite erstellen



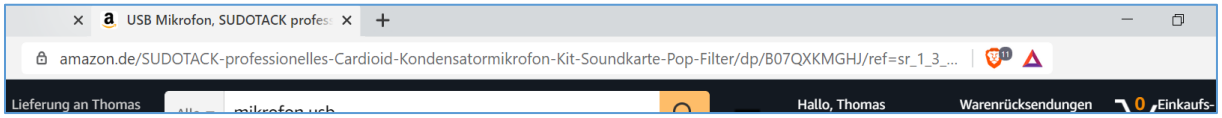
1. Route für Details erstellen
2. Template für die Produkt-Detail-Seite erstellen
3. Funktion für das Auslesen der Details erstellen
4. Route nach der Erstellung der Funktion anpassen
5. Sicherheitsschritte für falsche Eingaben in der URL erstellen

Der Klick auf den Button ist noch nicht eingerichtet.

1)Route erstelle

Dabei soll ein „SLUG“ verwendet werden. Das ist ein lesbarer Begriff in der URL, den man statt der Übergabe der ID verwendet. Damit kann man bessere Ergebnisse erzielen, wenn man mit google das Produkt sucht (SEO wird verbessert – search engine opimization).

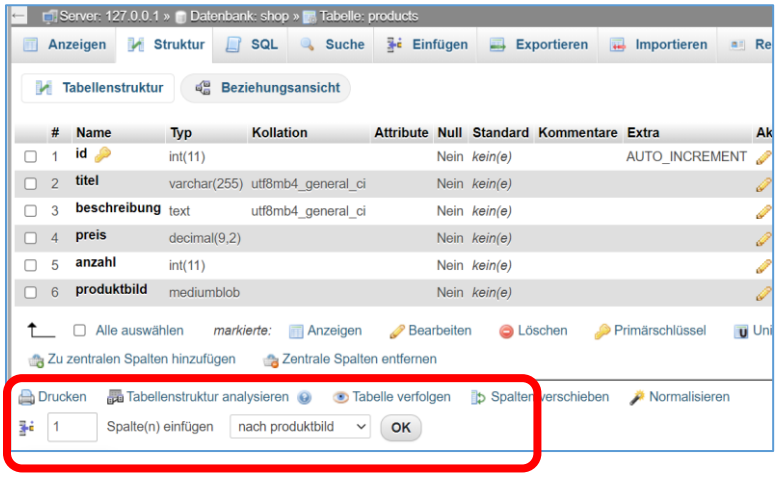
Beispiel bei amazon:



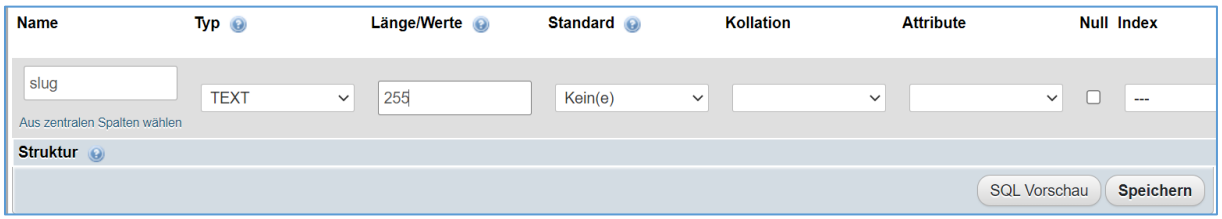
Normalerweise ist das auch der Titel, nur ohne Leerzeichen, die mit einem Verbindungsstrich ersetzt wurden.

1a)Datenbank Tabelle „products“ anpassen

Öffne die Datenbank und die Tabelle „products“ und füge ein neues Feld hinzu, das den Inhalt des „slug“ aufnimmt.



Der Name soll sein: slug und soll ein Text sein



Erstelle dafür einen „index“ – weil das bei der Suche schneller und besser wirkt

Haken setzten und dann auf „Index“ klicken. Fertig.

Nun muss man manuell ein paar Werte einbauen:

id	titel	beschreibung	preis	anzahl	produktbild	slug
1	5kg Orangen	ein kleiner Kübel mit Orangen	5.00	0	[BLOB - 60,9 KiB]	ein-kleiner-Kuebel
2	10 kg Orangen	süße 10 Kilo Orangen	8.00	0	[BLOB - 44,9 KiB]	suesse-10-Kilo
3	Ernte des ganzen Baumes	je nach Wachstum kann das sehr viel sein	99.00	0	[BLOB - 73,0 KiB]	Ernte-ganzer-Baum
4	drei Bäume gesamt	die Ernte von 3 Bäumen	249.00	0	[BLOB - 15,6 KiB]	Ernte-3-Baeume

Achtung: es dürfen

- **keine UMLAUTE und**
- **keine Sonderzeichen wie z.B. ß**

im Text vom „slug“ vorkommen. Ansonsten würde dann später nichts ausgegeben werden.

Fertig. Datenbank verlassen.

1b) Link einbauen in die Datei „card.php“

In „card.php“ liegt ja das Layout und der Inhalt für die Darstellung jedes Produkts als „card“.

Daher öffne „card.php“ und füge bei dem Button einen Link ein:

Dort ist der Link eine Weiterleitung auf die noch zu erstellende Route „index.php/product/“ mit dem Slug aus der Datenbank:

index.php/product/<?= $row['slug']?>

```
9     <div class="card-footer">
10        <a href="index.php/product/<?= $row['slug']?>" class="btn btn-primary
          ">Details</a>
11        <a href="index.php/cart/add/<?= $row['id']?>" class="btn btn-success ">In den
          Warenkorb</a>
```

Jedoch kann dieses „slug-Array“ nur benutzt werden, wenn es vorher, und das ist ja noch nicht erledigt, auch aus der Datenbank geholt wird. Hierfür muss man in der „product.php“ auch dieses Feld aus der Datenbank selektieren:

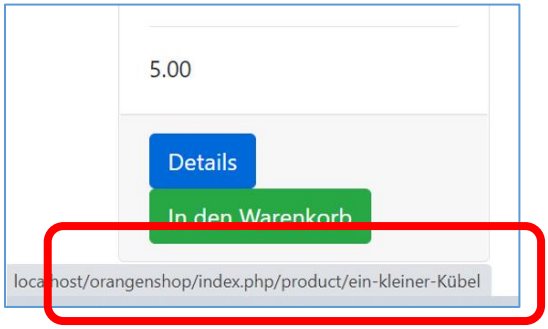
Öffne „porduct.php“ und suche die Funktion „getAllProducts“ und füge diese Zelle hinzu

```php
<?php

function getAllProducts(){
    $sql = "SELECT
    id,titel,beschreibung,preis,anzahl,produktbild, slug
    FROM products";
```

Fertig.

Nun steht dieser Array-Key zur Verfügung. Das kann man schon testen, indem man auf der Startseite mit der Maus über den Button „Detail“ drüberfährt um zu sehen, wie links (oder rechts unten - je nach Browser) der Pfad entsprechend schon vorhanden ist:



1c)Route in routes.php erstellen

Öffne die „routes.php“ und scrolle nach ganz unten, um eine neue Route anzulegen.

Erstelle das Grundgerüst für die neue Route „product“:

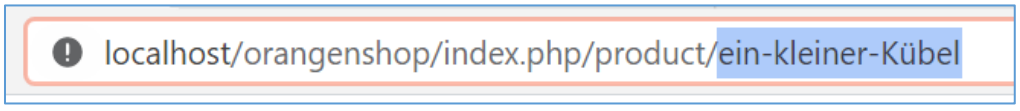
```
if(strpos($route,'/product') !== false) {

   exit();
}
```

```php
205
206 if(strpos($route,'/product') !== false) {
207
208     exit();
209 }
```

URL zerlegen:
Nun muss man sich den Teil aus der URL holen, der „slug“ genannt wurde, weil mir dieser verrät, um welches Produkt es sich handelt. Dafür verwendet man wieder die PHP Funktion „explode()“:

Gesucht ist dieser Teil der URL:

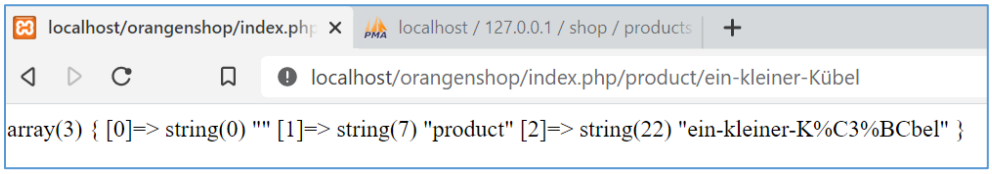


Diese Funktion haben wir schon ganz oben verwendet (in routes.php), wo wir ausgelesen haben, zu welchem Produkt man eins dazu oder ein weniger zählen soll:
Man könnte sich das von oben kopieren und dann ändern oder gleich neu schreiben:

$routeTeile = explode("/",$route);

```
206 ▼ if(strpos($route,'/product') !== false) {
207       $routeTeile = explode("/",$route);
```

Gibt man diese Variable aus in einem „var_dump()“ sieht man den Inhalt, um zu erkennen, dass es somit 3 „routeTeile“ gib und „der kleine Kübel“ an der Nummer 2 hängt. Diese ist eigentlich die Position 3, da man ja auch die 0 mitzählen muss.




```
206 ▼ if(strpos($route,'/product') !== false) {
207       $routeTeile = explode("/",$route);
208       var_dump($routeTeile);
```

Lösche den „var_dump“ wieder.

Nun kann man diese auslesen mit dem Code:

$slug = $routeTeile[2];

```
206 ▼ if(strpos($route,'/product') !== false) {
207       $routeTeile = explode("/",$route);
208       $slug = $routeTeile[2];
```

Füge nun gleich auch den passenden Endpunkt des Pfades ein, nämlich die Datei, die dann aufgerufen werden soll:

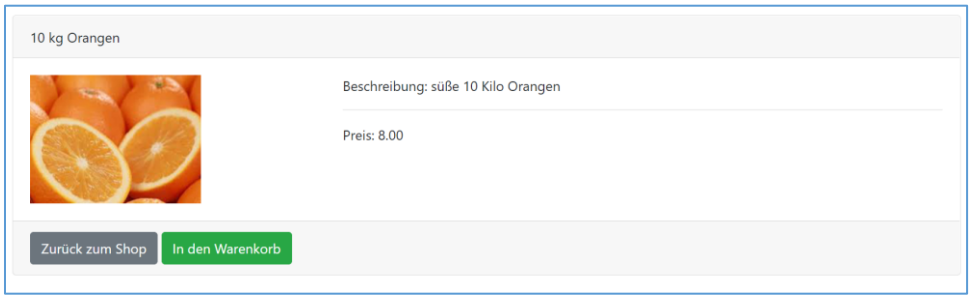
require_once __DIR__.'/templates/productDetails.php';

```
206 ▼ if(strpos($route,'/product') !== false) {
207       $routeTeile = explode("/",$route);
208       $slug = $routeTeile[2];
209
210       require_once __DIR__.'/templates/productDetails.php';
211
212       exit();
213   }
```

2)Template „productDetails.php“ erstellen

Erstelle dieses neue Template im Ordner „templates“.

Ziel: Das einzelne Produkt soll mit Bild schön angezeigt werden.

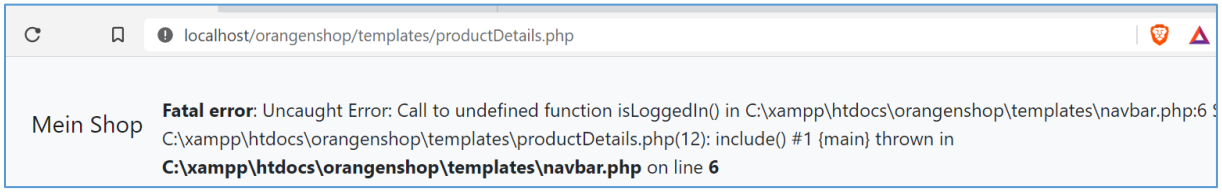


Dafür holen wir uns aus der „korbSeite.php“ wichtige Elemente, damit wir diese neue Seite nicht von Grund auf neu erstellen müssen. Nämlich alles von oben bis zum Ende des </header> und dann ganz unten die Einbindung des Scripts und das Ende vom body und html.

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <base href="/orangenshop/">
    <link rel="stylesheet" href="assets/css/bootstrap.css">
    <link rel="stylesheet" href="assets/css/styles.css">
    <title>Orangenshop</title>
  </head>
  <body>
    <?php include __DIR__.'/navbar.php' ?>
    <header class="jumbotron" >
    <div class="container" >
    <h1>Webshop für Orangen mit PHP</h1>
    </div>
    </header>


    <script src="assets/js/bootstrap.js"></script>
  </body>
</html>
```

Eine Kontrolle mit der Direkteingabe des Links beweist, dass es funktioniert, aber natürlich der Kontakt zu gewissen Elementen noch fehlt, da die Route das noch nicht vorgesehen hat:



Kein Problem, das ändert sich, nach der Einbindung und korrekten Route.

Weiterbearbeitung trotzdem.
Erstelle unter dem <header> eine neue <section>, darin einen „container“, damit es zentrierter ist und darin eine Klasse „card“ mit den einzelnen Elementen, so wie die Seite dann eben aussehen soll.

```
    </header>
    <section>
        <div class="container">
        <div class="card">
            <div class="card-header"></div>
            <div class="card-body">
            </div>
            <div class="card-footer">
            </div>
        </div>
        </div>
    </section>

    <script src="assets/js/bootstrap.js"></script>
```

```
<section>
<div class="container">
    <div class="card">
      <div class="card-header"></div>
      <div class="card-body"></div>
      <div class="card-footer"></div>
    </div>
 </div>
 </section>
```

Beginnen wir unten:

a)card-footer

- In den Footer der Card kommt der Button „In den Warenkorb“ den man aus der Datei „card.php“ kopieren kann. Dabei muss man aber die Variable ändern von $row auf die neue $productDetail, da diese in der Route ja angeboten wird.

- Zusätzlich soll ein Button „Zurück zum Shop“ angeboten werden.

Zurück zum Shop

```
<div class="card-footer">
  <a href="index.php" class="btn btn-secondary ">Zurück zum Shop</a>
  <a href="index.php/cart/add/<?= $productDetail['id']?>" class="btn btn-success
  ">In den Warenkorb</a>
</div>
```

b)card-header

- Im „card-header“ soll der Titel ausgegeben werden.

Info:
Die Werte, die man ausgeben möchte, kann man gleich **als PHP-Code** angeben. Dabei ist klar, dass das Befüllen aus der Datenbank erst in den nächsten Schritten hier erklärt wird.

Der PHP-Code wird üblicherweise mit „echo“ zur Ausgabe gebracht. Die Schnellschreibweise für
<?php echo…….?>
lautet
<?=……?>

Daher werden wir diese schnelle Codierung verwenden.

Hier wird die Variable verwendet, die in „routes.php“ angelegt wird. Diese greift dort auf die Funktion zu, die in der Datei „products.php“ angelegt wird.

<?= $productDetail['titel']?>

```
20          <div class="card">
21              <div class="card-header"><?= $productDetail['titel']?></div>
22              <div class="card-body">
```

c)card-body

In den „card-body“ kommen das Produktbild links und rechts daneben die Beschreibung und der Preis. Das soll, wie in Bootstrap üblich“ aufgeteilt werden, das Bild soll dabei 4 Spalten haben:

```
21              <div class="card-body">
22                  <div class="row">
23                      <div class="col-4"></div>
24                      <div class="col-8"></div>
25                  </div>
26              </div>
```

Im Body soll das Bild, die Beschreibung und der Preis ausgegeben werden, und zwar nebeneinander bzw. untereinander und getrennt durch eine dünne horizontale Linie <hr>.
Dazu verwendet man in Bootstrap die <col>-Möglichkeiten.

```
22              <div class="card-body">
23                  <div class="row">
24                      <div class="col-4"><?= $productDetail['produktbild']?></div>
25                      <div class="col-8">
26                          <div>Beschreibung: <?= $productDetail['beschreibung']?> </div>
27                           <hr>
28                          <div>Preis: <?= $productDetail['preis']?> </div>
29                      </div>
30                  </div>
31              </div>
```

```
<div class="row">
            <div class="col-4"><?= $productDetail['produktbild']?></div>
            <div class="col-8">
              <div>Beschreibung: <?= $productDetail['beschreibung']?> </div>
               <hr>
              <div>Preis: <?= $productDetail['preis']?> </div>
            </div>
          </div>
```

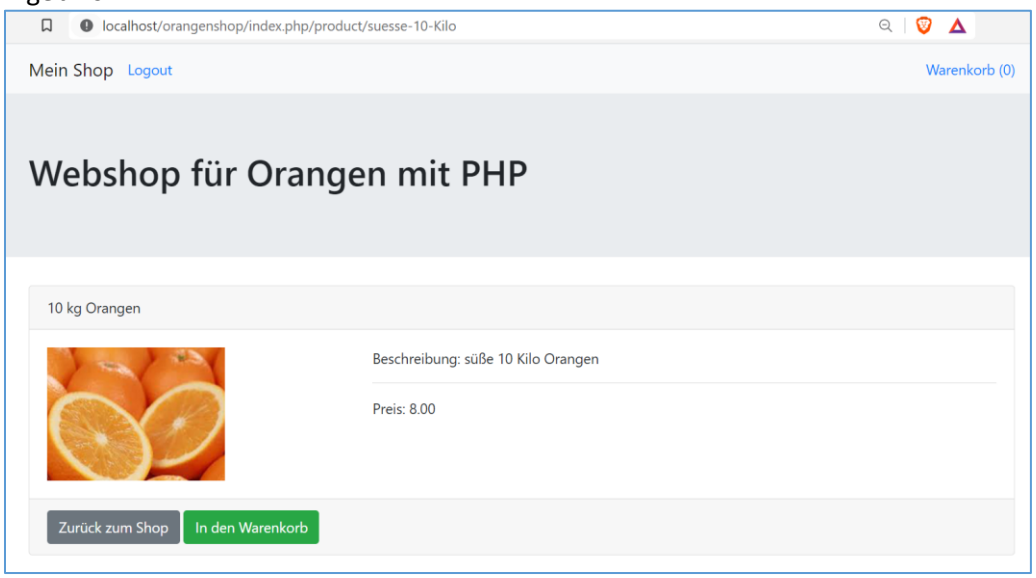
Dabei wird aber das **Produktbild nicht korrekt** dargestellt. Dafür muss man dieses noch bearbeiten:

Da wir diesen dafür nötigen Code schon in der „card.php“ verwendet haben, kann man sich dort bedienen:

```
<?php echo '<img height="150" width="200" src="'.base64_encode(
$productDetail['produktbild'] ).'"/>'; ?>
```

```
<div class="row">
    <div class="col-4">
    <?php echo '<img height="150" width="200"
    src="'.base64_encode( $productDetail['produktbild']
    ).'"/>'; ?>
    </div>
```

Ergebnis:



3)neue Funktion erstellen in „product.php“

Öffne die „product.php“ um eine neue Funktion zu erstellen, die die Details zum Produkt aus der Datenbank holt. Erstelle diese unterhalb der vorhandenen.

Diese Funktion erhält den „slug“, der ja ein String ist, daher als Parameter „string $slug“.

function getProductBySlug(string $slug){

}

```
12
13 ▼ function getProductBySlug(string $slug){
14
15   }
```

Als Antwort wird die Funktion ein ARRAY zurückgeben – daher der Doppelpunkt und array - mit den Produkteigenschaften bzw. Spalten aus der Datenbank. Darin wird die SQL-Abfrage definiert, die alle Elemente holt, die dem Produkt entsprechen, das mit dem SLUG übereinstimmt. Dabei soll aber nur ein Ergebnis gefunden werden – LIMIT 1.

```
13 ▼ function getProductBySlug(string $slug):array{
14       $sql = "SELECT id,titel,beschreibung,preis,anzahl,produktbild, slug
15       FROM products
16       WHERE slug=:slug
17       LIMIT 1";
```

Teile kann man sich von der Funktion darüber kopieren.

Dann wird das statement vorbereitet mit „prepare“:

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

```
13 ▼ function getProductBySlug(string $slug):array{
14       $sql = "SELECT id,titel,beschreibung,preis,anzahl,produktbild, slug
15       FROM products
16       WHERE slug=:slug
17       LIMIT 1";
18
19       $statement = getDB()->prepare($sql);
```

Dadurch wird schon in die Datenbank ein Befehl losgesendet, der mit dem Platzhalter :slug noch nichts anfangen kann. Daher muss man nun diesen aus der ersten Zeile aus dem Parametern übergeben, während man das statement ausführt (exekutiert):

$statement->execute(
  [':slug'=>$slug]
  );

```
17       LIMIT 1";
18
19       $statement = getDB()->prepare($sql);
20
21       $statement->execute(
22       [':slug'=>$slug]
23       );
```

Dann wird die komplette Spalte zurückgegeben.

return $statement->fetch();

```
19      $statement = getDB()->prepare($sql);
20
21      $statement->execute(
22      [':slug'=>$slug]
23      );
24
25      return $statement->fetch();
26  }
```

Sicherheitsabfragen einbauen:

Zur Sicherheit soll überprüft werden, ob das Statement erfolgreich war, wenn nicht soll ein „null“ returniert werden und nicht irgendwas anderes. Damit kann man Probleme abfangen, falls sich die Datenbank verändert hat oder ähnliches.

```
if(false === $statement){
    return null;
}
```

```
19      $statement = getDB()->prepare($sql);
20
21      if(false === $statement){
22          return null;
23      }
24
25      $statement->execute(
26      [':slug'=>$slug]
27      );
```

Dafür muss aber unbedingt auch in der ersten Zeile der Funktion ermöglicht werden, dass KEIN Array zurückgegeben wird, sondern, in diesem Fall (bei false) auch ein „null“. Daher muss vor das „array“ ein Fragezeichen gesetzt werden.

```
12
13  function getProductBySlug(string $slug):?array{
14      $sql = "SELECT id,titel,beschreibung,preis,anz
```

2. Sicherheitsabfrage

Was ist aber, wenn kein Produkt gefunden wird? Für den Fall, dass das die Antwort der „Exekution“ null wäre, wird mit der Funktion „rowCount()“ dieses ermittelt und dann, beim Eintreten von „0“ soll ebenfalls ein „return null“ zurückgegeben werden.
Füge das dort ein, bevor noch das Ergebnis „return“ stattfindet:

```
if($statement->rowCount() === 0){
    return null;
}
```

```
25      $statement->execute(
26      [':slug'=>$slug]
27      );
28
29      if($statement->rowCount() === 0){
30          return null;
31      }
32      return $statement->fetch();
33  }
```

Fertig.

4)Route anpassen, nachdem die Funktion erstellt wurde

Öffne die „routes.php“ und erstelle die Funktion, die die Produktdetails auslesen soll.

Erstelle eine neue Variable „$product“ und dieser wird ein Wert zugeordnet, der aus der eben erstellten Funktion geholt wird. Als Parameter wird der slug übergeben.

$productDetail = getProductBySlug($slug);

```
206 if(strpos($route,'/product') !== false) {
207      $routeTeile = explode("/",$route);
208      $slug = $routeTeile[2];
209
210      $productDetail = getProductBySlug($slug);
211
212      require_once __DIR__.'/templates/productDetails.php';
213
214      exit();
215  }
```

Diese Variable wird dann in der Datei „productDetails.php“ verwendet, um die Daten aus der Datenbank anzeigen zu können.

5)Sicherheitsschritte für falsche Eingaben in der URL erstellen

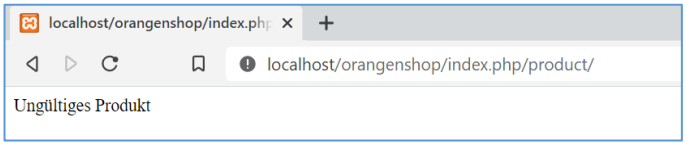
Folgendes soll mit einer entsprechenden Fehlermeldung abgefangen werden:

1. Jemand könnte in der URL den SLUG entfernen, um zu sehen was passiert
2. Jemand könnte den SLUG überschreiben mit einem nichtvorhandenen Produkt – es ist somit ein SLUG vorhanden, nur existiert dieser nicht

Dafür öffne die routes.php und erstelle entsprechende IF-Abfragen:

1)Kein SLUG vorhanden, da entfernt

Ziel: Fehlermeldung die passend ist



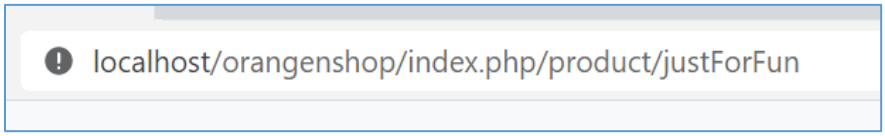
Unter der Variable $slug erstelle eine Abfrage, ob die Länge vom slug gleich Null ist – mit der Funktion „strlen()“ (string length)

```
if(0 === strlen($slug)){
    echo "Ungültiges Produkt";
    exit();
}
```

```
206 ▼ if(strpos($route,'/product') !== false) {
207       $routeTeile = explode("/",$route);
208       $slug = $routeTeile[2];
209
210 ▼     if(0 === strlen($slug)){
211           echo "Ungültiges Produkt";
212           exit();
213       }
```

2)Zwar ein SLUG vorhanden, aber kein Produkt dazu

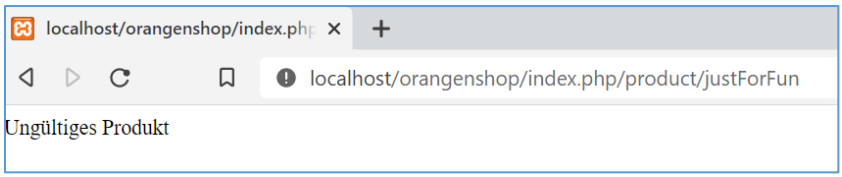
Blödsinn reinschreiben:



Dafür erstelle ebenfalls eine IF-Abfrage, die nachsieht, ob das ProduktDetail Null ist:

```
215       $productDetail = getProductBySlug($slug);
216
217 ▼     if(null === $productDetail){
218           echo "Ungültiges Produkt";
219           exit();
220       }
```

Ergebnis:



Quelle:
Vitalij Mik - youtube
https://www.youtube.com/watch?v=Tk5hpXiL4TY