OOP PHP

OOP PHP 1
1. BLOG OOP 3
1.1. INDEX.PHP 3
1.1.1. SIDEBAR.PHP 4
1.1.2. PAGINA: admin/users.php 4
1.2. DATABASE SETUP 4
1.2.1. TABEL: users 4
1.2.2. CONFIG.PHP en INIT.PHP 4
1.2.3. CLASSES 6
1.2.3.1. CLASS: DATABASE 6
1.2.3.2. CLASS: USER 10
1.2.3.3. CLASS: SESSION 20
1.2.4. ADMIN/LOGIN.PHP 24
1.2.5. ADMIN/LOGOUT.PHP 26
1.2.6. MESSAGE EN AUTOINVULLING 26
2. CRUD 28
2.1. CREATE 28
2.2. UPDATE 29
2.3. DELETE 30
3. ABSTRACTION VAN CODE 31
3.1. CRUD abstractie universele class 32
3.2. CRUD abstractie properties 33
4. PARENT CLASS 36
5. PHOTO CLASS 37
5.1. TABEL: photos 37
5.2. PHOTOS: CLASS TOEVOEGEN 38
5.3. PHOTOS: UPLOAD PATH EN PROPERTIES 39
5.4. PHOTOS: METHODES 40
5.5. PAGINA: UPLOAD 41
5.6. PAGINA: PHOTOS 42
5.6.1. PAGINA: PHOTOS WEERGEVEN 42
5.6.2. PAGINA: PHOTOS VERWIJDEREN 45
5.6.3. PAGINA: PHOTOS WIJZIGEN 49
5.7. PAGINA: USERS 56
5.7.1. PAGINA: USER VERWIJDEREN 59
5.7.2. PAGINA: USER TOEVOEGEN 59
5.7.3. PAGINA: USER WIJZIGEN 65

5.7.4. PAGINA: USER VERWIJDEREN 68
5.8. COMMENTS 69
5.9. COMMENT CLASS 69
5.10. COMMENTS FORMULIER 71
5.11. COMMENTAAR MAKEN/TONEN IN DE FRONT-END 71
5.12. COMMENTAAR MAKEN/TONEN IN DE BACK-END 81
5.12.1. COMMENTAAR DELETEN IN DE BACK-END 82
6. ADMIN DASHBOARD 86
6.1. DASHBOARD SNIPPETS 86
6.2. GOOGLE API CHARTS 90
6.3. PAGE VIEWS 91
6.4. UNIVERSELE COUNT METHODE 92
7. FRONT-END: index.php 94
8. PAGINEREN 95
9. HOSTING 98
10. EXTRA UITLEG OVER =FILES 99
10.1. FILE TOEGANG 99
10.2. FILE UPLOAD 99
10.2.1. FILE STRUCTUUR VOOR UPLOAD 100

1. BLOG OOP

https://startbootstrap.com/themes/sb-admin-2/
Download de Admin UI SB ADMIN 2 en maak op je localhost een nieuwe map aan met de naam blog .
Unzip het bestand en kopieer alles onder de map blog .

We gaan de structuur van de index.php van het admin gedeelte nog iets verder aanpassen. De bedoeling hier is dat we de index.html pagina eerst zullen strippen naar php bestanden.

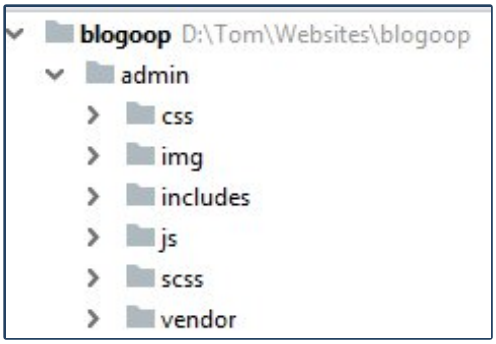
1.1. INDEX.PHP

Aangezien we niet de frontend maar de user interface van het backend gedeelte maken creëren we in de root van ons project een mapje aan: **admin**. Daarin maken we nog een map nl. **includes**.
De index.html pagina kopieer je met de rest van de mapjes onder het nieuwe admin mapje.

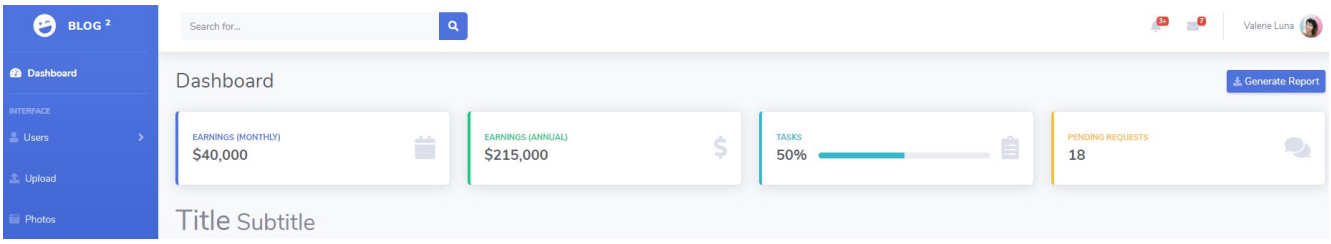
Hernoem dan als eerste stap ook hier index.html naar index.php.

Surf nu naar de localhost locatie waar de index pagina in het admin gedeelte staat.
Afhankelijk vanje installatie kan dit het volgende zijn:
localhost/blog/admin/index.php

Resultaat:



Zorg ervoor dat we nu reeds het volgende resultaat te zien krijgen. Pas hiervoor de bestanden aan waar nodig.



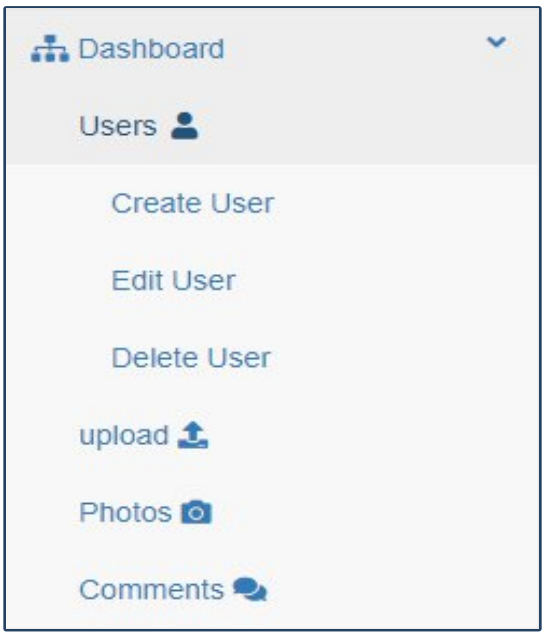
Je dient dus de pagina te strippen:

Maak de volgende pagina's aan en verdeel de inhoud van admin/index.php als volgt:

```
<?php include("includes/header.php"); ?>
<?php include("includes/sidebar.php"); ?>
<?php include("includes/content-top.php"); ?>
<?php include("includes/content.php"); ?>
<?php include("includes/footer.php"); ?>
```

1.1.1. SIDEBAR.PHP

We passen voorlopig de sidebar aan. Het is mogelijk dat je de versie van fontawesome in header.php dient te wijzigen naar de laatste versie om de icoontjes toe te voegen. Met jullie kennis van bootstrap en html moet dit zeker lukken!



1.1.2. PAGINA: admin/users.php

Kopieer alles uit index.php en plak dit in de users.php pagina. We passen dit aan naar onze noden.

Kopieer de users.php pagina 3x en hernoem deze respectievelijk in comments.php, photos.php en upload.php

1.2. DATABASE SETUP

Maak een nieuwe database aan met de naam **dbblogoop** in phpmyadmin.

1.2.1. TABEL: users

Maak een tabel users aan met de volgende velden:

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1	**id**	int(11)			No	*None*		AUTO_INCREMENT
☐	2	**username**	varchar(255)	utf8_general_ci		No	*None*		
☐	3	**password**	varchar(255)	utf8_general_ci		No	*None*		
☐	4	**first_name**	varchar(255)	utf8_general_ci		No	*None*		
☐	5	**last_name**	varchar(255)	utf8_general_ci		No	*None*		

1.2.2. CONFIG.PHP en INIT.PHP

In admin/includes voeg je een **config.php** pagina toe die de databaseconnectie zal bevatten. De functie define definieert een CONSTANTE over de volledige applicatie heen.

```php
<?php
    define('DB_HOST', 'localhost');
    define('DB_USER', 'root');
    define('DB_PASS', '');
    define('DB_NAME', 'dbblogoop');
?>
```

In admin/includes voeg je een nieuwe pagina toe, nl. init.php. Deze pagina zal alle includes bevatten die we gaan gebruiken in ons project. Een eerste include zal de config.php worden:

```php
<?php include("config.php"); ?>
```

Wanneer onze admin/index.php wordt geladen, dan wordt header.php eerst geladen. We dienen dus een include in de header te voorzien die verwijst naar init.php die op zijn beurt verwijst naar config.php met de database connectie.
Open admin/header.php en pas aan als volgt:

```php
<?php include("init.php"); ?>
<!DOCTYPE html>
<html lang="en">

<head>
```

1.2.3. CLASSES

1.2.3.1. CLASS: DATABASE

Maak in admin/includes een nieuwe class aan, nl: Database.php

```php
<?php
/** Created by PhpStorm. ...*/
require_once("config.php");
class database
{
    public $connection;
    function __construct(){
        $this->open_db_connection();
    }

    public function open_db_connection(){
        $this->connection = mysqli_connect( host: DB_HOST, user: DB_USER, password: DB_PASS, database: DB_NAME);
        if(mysqli_connect_errno()){
            die("Database connection mislukt" .mysqli_error());
        }
    }
}
$database = new Database();
?>
```

Merk op dat we de de database constanten die zich in config.php bevinden hier in deze class inladen met require_once i.p.v. include.
Wat zijn nu de verschillen?

Include:
probeert op de webserver het proces toch verder te verwerken ook al is het bestand die het zoekt fysisch niet aanwezig op de webserver.
Require:
Stopt het proces als het bestand niet beschikbaar zou zijn.
Once
Dit werk zowel bij include_once als require_once. Hiermee vermijden we dat het bestand meer dan 1 keer kan worden opgevraagd.

De class Database bevat een public variabele $connection. Public impliceert dat de variabele ook buiten de class aanspreekbaar is.
<u>methode open_db_connection</u>
De class Database krijgt tevens een methode (functie) waarin we de built-in functie mysqli_connect aanspreken, die de variabelen bevat om een connectie te kunnen maken me onze database dbblogoop.
Merk op dat het keyword $this slaat op de class Database ZELF.
We testen tevens of er een error wordt gegenereerd door phpmyadmin en geven in dat geval een error terug op het scherm d.m.v. mysqli_error.
De standaard constructor van de class database is function __construct en voert automatisch alles van code uit die hier staat, telkens de class wordt aangeroepen. We willen namelijk een automatische opening van de connectie open_db_connection() die zorgt voor de connectie met onze database.

Voeg in **init.php** de class Database.php toe:

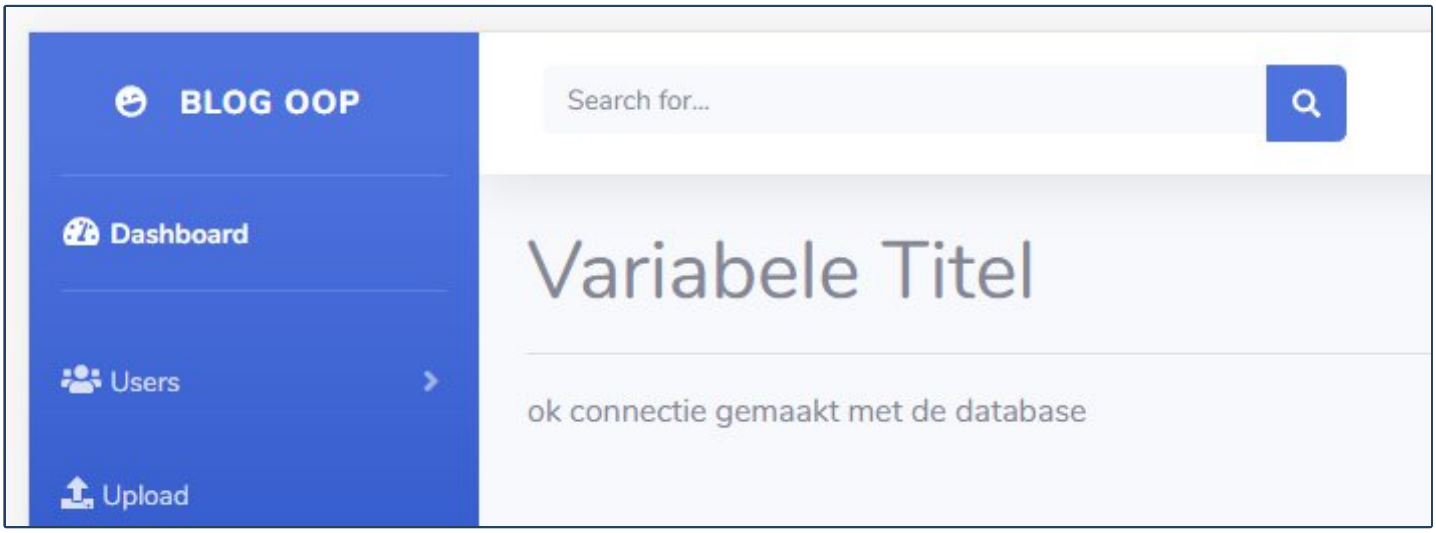
```php
<?php
include("config.php");
include("database.php");
?>
```

Open nu admin/content.php
Kopieer vervolgens de volgende inhoud uit admin/index.php en plak deze erin.
Zoals je hieronder kan zien testen me meteen of er een connectie kan worden gemaakt met onze database.

```
<!-- Begin Page Content -->
<div class="container-fluid">
    <div class="row">
        <div class="col-12">
                    <h1 class="page-header">
                      Variabele Titel
                    </h1>
                    <hr>
                    <?php
                        if($database->connection){
                            echo "ok connectie gemaakt met de database";
                        }
                    ?>
        </div>
    </div>
</div>
<!-- /.container-fluid -->
</div>
<!-- End of Main Content -->
```

Surf nu naar localhost/blog/admin/index.php pagina en test de connectie uit!

Resultaat:



De volgende methode binnen de class Database zullen deze zijn voor het uitvoeren van queries.
methode query($sql)
Deze methode maakt gebruik van 2 parameters nl, de connectie zelf en de sql string.

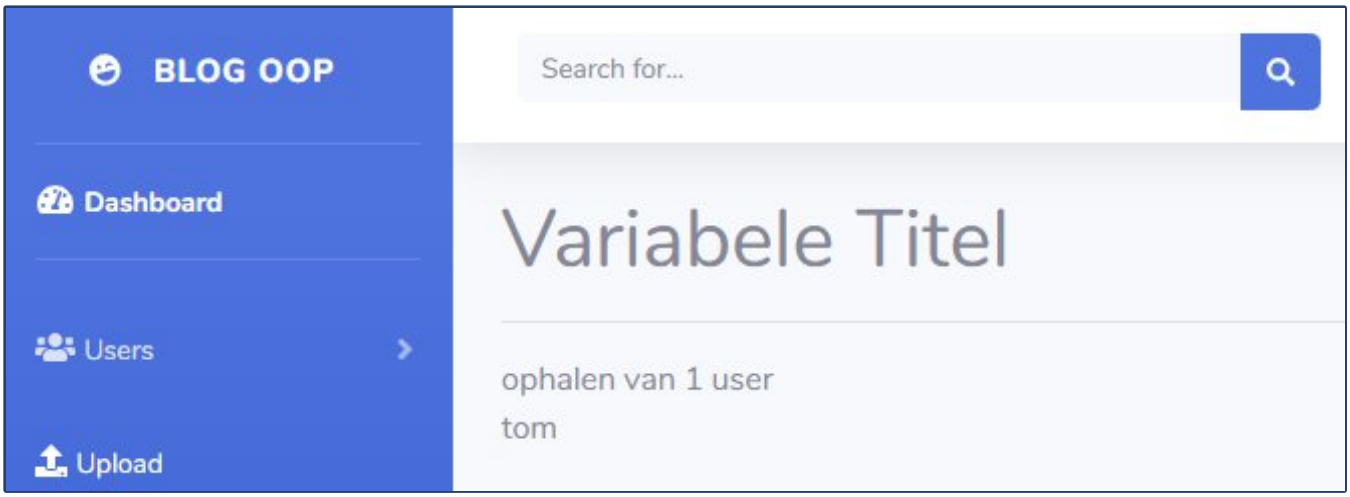
```php
public function open_db_connection(){
    $this->connection = mysqli_connect( host: DB_HOST, user: DB_USER, password:
    if(mysqli_connect_errno()){
        die("Database connection mislukt" .mysqli_error());
    }
}

private function confirm_query($result){
    if(!$result){
        die("query kon niet worden uitgevoerd" .$this->connection->error);
    }
}

public function query($sql){
    $result = $this->connection->query($sql);
    $this->confirm_query($result);
    return $result;
}

public function escape_string($string){
    $escaped_string = $this->connection->real_escape_string($string);
    return $escaped_string;
}

atabase = new Database();
```

De functie real_escape_string zorgt ervoor dat vreemde karakters uit sql queries worden verwijderd. Dit vermijd o.a. SQL INJECTIONS.

We controleren of bovenstaande queries werken. Hiervoor hebben we data nodig in de database.
Voeg enkele users handmatig toe in de database.
Pas admin/includes/content.php aan om dit testen als volgt:
we vragen hier de object variabele database op van de class, op die manier hebben we toegang tot alle methodes die we daarnet hebben geschreven. ¶
Dit is de procedurele manier van schrijven.¶

```php
</h1>
<?php
    echo "ophalen van 1 user" . "<br>";
    $sql = "SELECT * FROM users WHERE id=1";
    $result = $database->query($sql);
    $user_found = mysqli_fetch_array($result);
    echo $user_found["username"];
?>
```

Resultaat:


BLOG OOP
Search for...
Dashboard
Variabele Titel
Users
ophalen van 1 user
tom
Upload

1.2.3.2. CLASS: USER¶

Bovenstaande query toonde u de procedurele manier van schrijven.¶
Wanneer we echter data opvragen, in dit geval de users van een database dan gebruiken we hiervoor classes. We maken dus een object van groep van users, dit noemt men OOP programmeren.
We willen een query uitvoeren van al onze users.
Hiervoor hebben we de methode query() geschreven die we in onze database class hebben geschreven. Om deze methode te benaderen dienen we een globale variabele $database te creëeren.

```php
<?php
class User{

    public function find_all_users(){
        global $database;
        $result = $database->query( sql: "SELECT * FROM users");
        return $result;
    }
}

?>
```

¶
¶

Nu kunnen we bovenstaande code testen:
om de class user overal kenbaar te maken dienen we deze eerste toe te voegen aan init.php

```php
<?php
include("config.php");
include("Database.php");
include("User.php");
?>
```

In admin/includes/content.php wijzig je als volgt:
We maken een nieuwe object variabele $user van de class User().
Vervolgens spreken we de methode find_all_users() aan van deze class waarvan het restulaat zich in de variabele $result zal bevinden.
Deze variable loopen we vervolgens om de data te doorlopen.¶
¶
Manier 1

```
<div class="col-12">
            <h1 class="page-header">
              Variabele Titel
            </h1>
            <hr>
            <?php
                $user = new User();
                $result = $user->find_all_users();
                while($row = mysqli_fetch_array($result)){
                    echo $row['username'] . "<br>";
                }
            ?>
</div>
```

Hierboven dienen we eerst een object te creëeren ($user = new User();). Bovenstaande is reeds zeer goede OOP code maar we kunnen dit nog korter schrijven.¶
¶
Manier 2: STATIC
De statische manier van het gebruik van classes binnen OOP is de manier om volgens de laatste standaarden te schrijven. Dit wordt zo toegepast in frameworks zoals LARAVEL die we later van nabij zullen bekijken.

```
</h1>
<?php

 $result = User::find_all_users();
 while($row = mysqli_fetch_array($result)){
    echo $row['username'] . "<br>";
 }
?>
```

Om bovenstaande te kunnen gebruiken dienen we enkel de functie als static te beschouwen. Open admin/includes/User.php en pas aan als volgt:

```
class User
{
    public static function find_all_users(){
        global $database;
        $result = $database->query( sql: "SELECT * FROM users");
        return $result;
    }
}
```

methode find_user_by_id

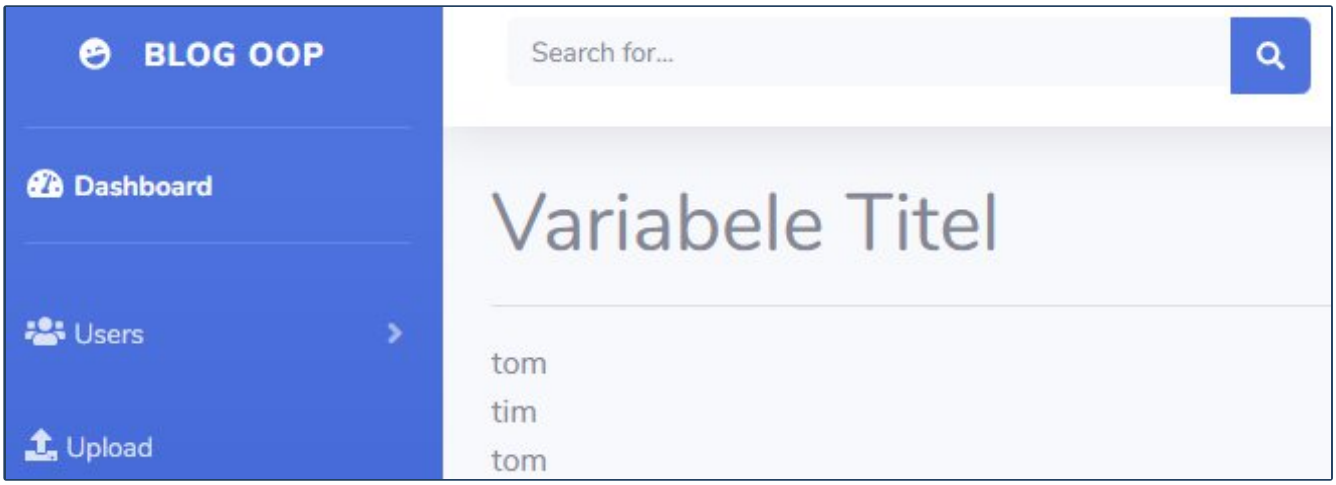
In admin/includes/User.php voeg je de volgende methode toe. Deze methode zoekt de user in de database via zijn id.

```php
public static function find_user_by_id($user_id){
    global $database;
    $result = $database->query( sql: "SELECT * FROM users WHERE id=$user_id");
    $user_found = mysqli_fetch_array($result);
    return $user_found;
}
```

Test dit terug in admin/includes/content.php en geef de username op het scherm terug.

```php
<?php
   $result = User::find_all_users();
   while($row = mysqli_fetch_array($result)){
       echo $row['username'] . "<br>";
   }

   $result = User::find_user_by_id( user_id: 1);
     while($row = mysqli_fetch_array($result)){
         echo $row['username'] . "<br>";
     }
?>
```

Resultaat:



We kunnen de laatste queries nog vereenvoudigen door een method te gaan schrijven die iedere query kan uitvoeren. In de voorgaande queries zag je nog de object variabele $database met de methode query() van de class database die werd aangeroepen. **($database->query()).**
Wanneer we dit gedeelte in een aparte methode gieten (**find_this_query**) in de class user zelf dan kunnen we find_this_query binnen de andere methodes van de classe aanspreken in combinatie met het **self** keyword.
self::find_this_query().

```php
class User
{
    public static function find_this_query($sql){
        global $database;
        $result = $database->query($sql);
        return $result;
    }

    public static function find_all_users(){
        return self::find_this_query( sql: "SELECT * FROM USERS");
    }

    public static function find_user_by_id($user_id){
      $result = self::find_this_query( sql: "SELECT * FROM USERS WHERE id= $user_id");
      $user_found = mysqli_fetch_array($result);
      return $user_found;
    }
}
```

Arrays wijzigen naar objecten
Wanneer we kijken naar admin/includes/content.php dan retourneren we steeds arrays. We willen echter geen arrays retourneren maar objecten in OOP.
In OOP schrijven we dit bijvoorbeeld als
$user->username,
$user->id,
$user->first_name,...

Om dit te doen werken dienen we in de User class de volgende variabelen toe te voegen.

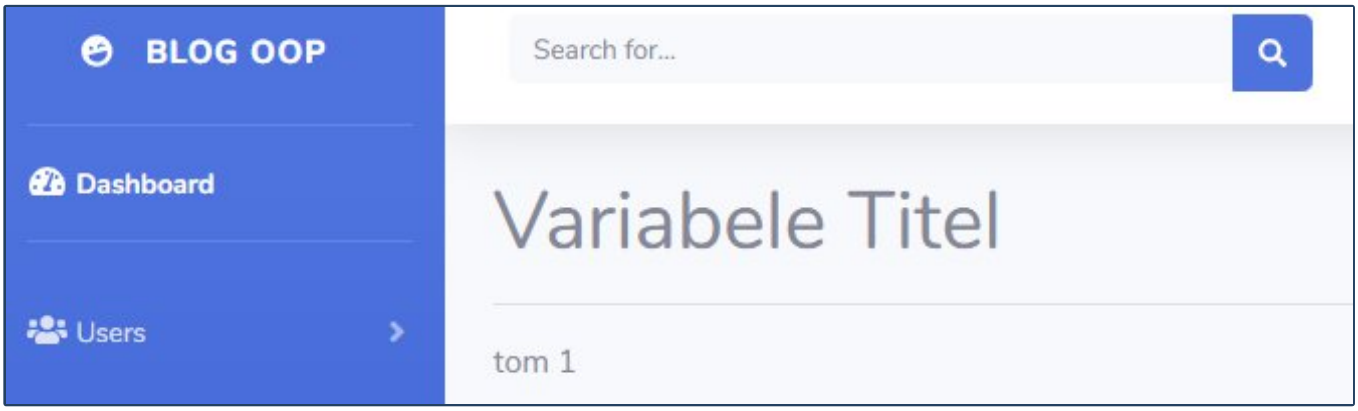
```php
<?php
/** Created by PhpStorm. ...*/

class user
{
    /*** PROPERTIES ***/
    public $id;
    public $username;
    public $password;
    public $first_name;
    public $last_name;
```

In admin/includes/content.php dienen we de volgende code neer te schrijven om dit als object te gaan schrijven in een webpagina:
zoals je kan zien wordt in onderstaand voorbeeld de id als object afgedrukt op het scherm.

```
<div class="col-12">
            <h1 class="page-header">
              Variabele Titel
            </h1>
            <hr>
            <?php
             /* $result = User::find_all_users();
             while($row = mysqli_fetch_array($result)){
                 echo $row['username'] . "<br>";
             }*/

             $result = User::find_user_by_id( user_id: 1);
             $user = new User();
             $user->id = $result['id'];
             $user->username = $result['username'];
             /** momenteel maken we nog geen gebruik van password hashing*/
             $user->password = $result['password'];
             $user->first_name = $result['first_name'];
             $user->last_name = $result['last_name'];
             echo $user->username . ' ' . $user->id;
          ?>
</div>
```

Test dit uit, je zal naam en userid op het scherm zien staan.

Resultaat:



Bovenstaande code gaan we nog iets vereenvoudigen door ook hier een stuk ervan in een method te gieten **in de user class.**
Knip enkel het gedeelte van het object weg, laat dus de eerste lijn $result staan .
We plakken dit in een nieuwe functie in de user class. Wijzig alle $user variabelen naar $the_object variabele. De bedoeling van deze methode is om ervoor te zorgen dat we gelijk welk veld die later nog wordt toegevoegd gemakkelijk aan de code kunnen toevoegen in deze methode. Aangezien de function static is kunnen we deze als new **self** declareren.

```php
public static function instantie($result){
    $the_object = new self();
    $the_object->id = $result['id'];
    $the_object->username = $result['username'];
    $the_object->password = $result['password'];
    $the_object->first_name = $result['first_name'];
    $the_object->last_name = $result['last_name'];
    return $the_object;
}
```

In **admin/includes/content.php** kun je nu de code als volgt gaan gebruiken: in $result zit het volledige resultaat van de query. Deze object variabele gebruiken we om ons object via de methode instantie in de User class te vullen.
De variabele $user bevat bijgevolg het volledige gevulde object die we op een eenvoudige manier kunnen afdrukken op het scherm.

```php
$result = User::find_user_by_id( user_id: 1);
$user = User::instantie($result);
echo $user->username . ' ' . $user->id;
```

Velen onder jullie stellen zich nu de vraag: "Indien ik per tabel nu elk 100 velden heb, moet ik deze dan op die manier declareren?". Gelukkig is dit niet zo, dit lossen we op in het volgende stuk door terug onze code te vereenvoudigen.

In onze functie instantie van de User class gaan we met een foreach door de objecten lussen (loopen).

<u>Attributen en waarden van object inlezen en weergeven</u>

Het niveau van OOP schrijven binnen de pagina's zal uiteindelijk in het volgende moeten resulteren:

```php
<div class="col-12">
            <h1 class="page-header">
              Variabele Titel
            </h1>
            <hr>
            <?php
             /* $result = User::find_all_users();
             while($row = mysqli_fetch_array($result)){
                 echo $row['username'] . "<br>";
             }*/

             $result = User::find_user_by_id( user_id: 1);
             $user = User::instantie($result);
             echo $user->username . ' ' . $user->id;

          ?>
            <hr>
            <?php
                $users = User::find_all_users();
                foreach($users as $user){
                    echo $user->username;
                }
            ?>
</div>
```

Bovenstaande **kan nu nog niet werken** aangezien we objecten loopen!

We spreken de methode **find_all_users** aan uit de **class User**. Daarna lussen we door alle users. OPMERKING: **we lussen hier door objecten! M.a.w. vooraleer we onderstaande kunnen gebruiken moeten we zowel de velden alsook hun waarden per record uit een tabel kunnen lezen.**

Om bovenstaande te bereiken gaan we de functies aan passen in de User class:
find_all_users zorgt ervoor dat we een sql string retourneren en overdragen naar de functie **find_this_query** zoals je hieronder kan zien. We willen nu een array maken van objecten en dit declareren we door een variabele **$the_object_array** te maken. Deze array wordt gevuld d.m.v. de onderstaande lus. De $result_set wordt overgedragen naar $the_object_array d.m.v. onze instantantion. Aangezien instantantion een static methode (functie) is, kunnen we hier meteen het keyword **self** gebruiken. We geven meteen een parameter $row mee die het resultaat is van de functie instantie .

```
/*** METHODS ***/
public static function find_this_query($sql){
    global $database;
    $result = $database->query($sql);
    $the_object_array = array();
    while($row = mysqli_fetch_array($result)){
        $the_object_array[] = self::instantie($row);
    }
    return $the_object_array;
}
```

Wanneer we kijken welke waarden er worden overgedragen door instantie ($row), dan dienen we deze even van naderbij te bekijken. Pas als volgt aan. (zien onderstaande screenshots)

We benaderen $result die als parameter is binnengekomen in bovenstaande functie als een associatieve array. D.w.z. dat we zowel de sleutel (veld) en waarde lussen.
In de foreach zie je een nieuwe methode **has_the_attribute** staan die we eerst van naderbij bekijken. Deze methode ontvangt uiteindelijk een parameter uit onze foreach loop. Deze **parameter is $the_attribute.** Het doel van de has_the_attribute methode is om na te kijken of er wel degelijk (attributen) in de tabel aanwezig zijn. Het attribuut is uiteindelijk de sleutel van een associatieve array.
In ons voorbeeld: (result as **$the_attribute** => value)
De variabele **$object_properties** wordt gevuld met alle variabelen van de User class die we bovenaan onze user class hebben gedeclareerd. Hiervoor gebruiken we de handige built-in functie van php, nl. get_object_vars($this). De parameter **$this** verwijst dus naar de classnaam waar hij zich in bevindt, met name User.
De uiteindelijke return van de functie halen we op met de built-in functie **array_key_exists** met 2 parameters, nl. $the_attribute en $object_properties.
Deze functie geeft een returnwaarde van true of false terug. We controleren of $the_attribute parameter aanwezig is in de array variabele $object_properties.

Wanneer dit true is dan wordt de waarde toegekend aan het attribuut van het object.
($the_object->$the_attribute = $value).

```php
public static function instantie($result){
    $the_object = new self;

    foreach($result as $the_attribute => $value){
        if($the_object->has_the_attribute($the_attribute)){
            $the_object->$the_attribute = $value;
        }
    }

    return $the_object;
}
```

```php
private function has_the_attribute($the_attribute){
    $object_properties = get_object_vars($this);

    return array_key_exists($the_attribute, $object_properties);
}
```

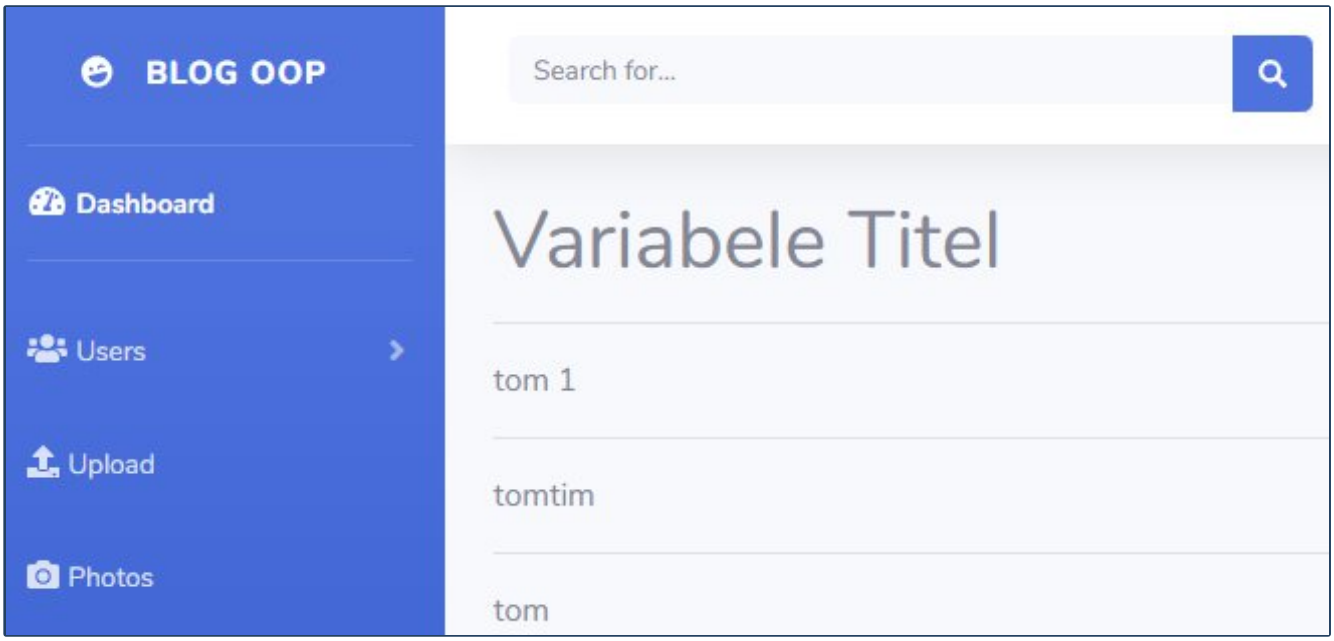
Gebruik de instantantie method om 1 user te zoeken.
We schrijven if/else statement in itenary notation (shorthand).
In het **true** gedeelte (**array_shift($the_result_array)**) zorgt deze functie ervoor dat het eerste item wordt uitgelezen.

```
public static function find_user_by_id($user_id){
    $result = self::find_this_query( sql: "SELECT * FROM users WHERE id=$user_id");
    return !empty($result) ? array_shift( &array: $result) : false;
}
```

In content.php pas je als volgt aan om bovenstaande te testen.

```
$result = User::find_user_by_id( user_id: 1);
echo $result->username;
```

Resultaat:



Autoload van alle classes
Tot nu toe moesten we ieder nieuw gemaakte class ook initialiseren in init.php.
Dit gaan we automatiseren door een autoload te schrijven die alle classes uit je project automatisch inleest, zodat we deze niet telkens in init.php dienen te schrijven.

Maak een **functions.php** bestand aan in admin/includes en voeg de volgde code in:
Er bestaat een mogelijkheid om classes automatisch uit een map te lezen. We gaan dus controleren of er class bestanden op een bepaalde locatie (path) staan. In ons project zit onze User class op het path admin/includes/User.php
Wanneer we nu een functie **classAutoLoader** schrijven zal deze het bestand User.php vinden. Daarna. zorgen we ervoor dat we het bestand gaan includen (of require_once). Je kan meerdere classes van autoloaders maken indien je wenst. De registratie van deze classes doet men met de built-in functie **spl_autoload_register().**

```php
<?php
    function classAutoLoader($class){
        $class = strtolower($class);
        $the_path = "includes/{$class}.php";

        if(is_file($the_path) )

        if(file_exists($the_path) && !class_exists($class)){
            include($the_path);
        }
        else{
            die("This file name {$class}.php was not found!");
        }
    }

 spl_autoload_register( autoload_function: 'classAutoLoader');
?>
```

D.w.z. dat we ons **init.php** bestand kunnen aanpassen: we doen een include van functions.php en zetten User.php in commentaarlijnen. M.a.w. functions.php zal zelf zoeken in admin/includes of de class User.php bestaat.

```php
include("functions.php");
include("config.php");
include("Database.php");
//include("User.php");
```

Het programma zou nu dus nog steeds moeten werken. Probeer maar uit!

SAMENVATTING USER CLASS

De bedoeling van de code op deze manier te schrijven is het creëeren van je eigen API om universele code te schrijven.

Bekijk hieronder nog eens de stappen wat de User class nu eigenlijk doet:

User::find_all()

1. Hiermee gaan we naar de find_all() methode.
2. Find_all() retourneert find_by_query() resultaten.
3. Find_by_query():
    - maakt de query
    - haalt de data van de tabel op met een while loop en retourneert $row
    - het resultaat van $row wordt overgedragen naar instantantion methode.
    - retourneert het object in de $the_object_array variabele

Wat doet de instantantion methode?

1. Grijpt de class name die wordt opgeroepen
2. Creëert een instantie van de class.
3. Er wordt door de $the_record variabele gelust die alle records bevat
4. het kijkt na of er properties bestaan binnen het object met de methode has_the_property() of has_the_attribute()
5. wanneer de properties/attributes (keys) worden gevonden dan wordt de waarde toegekend.
6. De return is het volledige object.

~~1.2.3.2.~~1.2.3.3. CLASS: SESSION

Maak een nieuwe pagina aan admin/includes/**Session.php**
Met deze classe willen nagaan of een user is ingelogd of niet. In het geval hij ingelogd is dat gaan wij hem naar admin/includes/index.php brengen in het andere geval naar de index.php pagina van de publieke kant.

Om dit stuk te starten, zorg ervoor dat **include("User.php")** in init.php terug **actief** staat en pas als volgt aan. We maken even geen gebruik van de autoload mogelijkheid. Pas als volgt aan:

```php
<?php
/**
 * Created by PhpStorm.
 * User: user
 * Date: 6/02/2019
 * Time: 10:12
 */
require_once("functions.php");
require_once("config.php");
require_once("Database.php");
require_once("User.php");
require_once("Session.php");

?>
```

require_once zal ons een error geven. Include geeft ons enkel een warning. Wanneer er iets misloopt willen we wel degelijk de error zien verschijnen.

Doe dit voor admin/includes/header.php ook:

```php
<?php require_once("init.php"); ?>

<!DOCTYPE html>
<html lang="en">
```

Open nu admin/includes/Session.php
Deze classe schrijven we om sessies van users **automatisch** te laten starten. Hieronder doen we dit natuurlijk met de reeds gekende **functie __construct** die dankzij het maken van de nieuwe instantie **($session = new Session())** een session id start.

```php
<?php
class Session{

    private $signed_in;
    public $user_id;

    /*automatisch starten van een session*/
    function __construct(){
        session_start();
    }

}
$session = new Session();
?>
```

Nu dienen we na te kijken of het userid wel degelijk werd geset door de sessie.
we schrijven hiervoor een functie **check_the_login()**, die we ook automatisch laten starten in __construct().
We starten met de **$signed_in** variabele op **false** te zetten als default. False staat voor niet ingelogd.
Wanneer de public variabele $user_id inderdaad als session variabele is geset, dan kennen we de session variabele toe aan het session object
($this->user_id = $_SESSION['user_id']);
Tegelijk zetten we de object variabele **signed_in** op **true**.
In het else statement gedeelte gaan we de variabele gaan "ledigen" = **unset**, wanneer er geen session variabele wordt gevonden.
Logisch gezien zetten we dan de variabele **signed_in** op **false**.

```php
<?php
class Session{

    private $signed_in = false;
    public $user_id;

    /*automatisch starten van een session*/
    function __construct(){
        session_start();
        $this->check_the_login();
    }
    /*check: is de session userid geset!*/
    private function check_the_login(){
        if(isset($_SESSION['user_id'])){
            $this->user_id = $_SESSION['user_id'];
            $this->signed_in = true;
        }
        else{
            unset($this->user_id);
            $this->signed_in = false;
        }
    }
}
$session = new Session();
?>
```

<u>Login methode.</u>
Dit is een methode die een private variabele van de class kan lezen. Die gebruiken we om na te kijken of de user is ingelogd of niet. De methode login($user) zal dus in de parameter $user een user ontvangen. Indien dit waar (true) is dan zullen we het session id toekennen aan een object variabele.

```
$this->user_id = $_SESSION['user_id'];
$this->signed_in = true;
```

De methode is_signed_in zal enkel een boolean teruggeven,nl. true of false.
Deze 2 methodes zullen we dus later nodig hebben bij het inloggen.

```php
    /*automatisch starten van een session*/
    function __construct(){
        session_start();
        $this->check_the_login();
    }

    public function is_signed_in(){
        return $this->signed_in;
    }

    public function login($user){
        if($user){
            $this->user_id = $_SESSION['user_id'] = $user->id;
            $this->signed_in = true;
        }
    }
```

Logout methode.
In de logout methode unsetten we de session id en de user id. Daarnaast gaan we de object variabele signed_in terug op false zetten.

```php
public function logout(){
    unset($_SESSION['user_id']);
    unset($this->user_id);
    $this->signed_in = false;
}
/*check: is de session userid geset!*/
private function check_the_login(){
```

Inloggen op includes/admin/index.php
Wanneer we in de backend willen inloggen, dan loggen we in op index.php. Index php wordt o.a. samengesteld uit **header.php.**
Header.php is het eerste gedeelte die wordt geladen en per definitie de beste plaats om te **testen** of er een **sessie actief** is!

Open index.php en pas als volgt aan:
!session = het omgekeerde van een sessie, m.a.w. wanneer we geen user vinden die een sessie heeft.
In dit geval voeren we de method **is_signed_in** uit.
Deze user heeft dus geen toegang en wordt **geredirect** naar **login.php** die we nog dienen te schrijven!

```php
<?php include("includes/header.php"); ?>
<?php
if(!$session->is_signed_in()){
    redirect( location: "login.php");
}
?>
```

De **functie redirect** is geen standaard functie, maar een zelfgeschreven functie. Open functies.php en pas als volgt aan:

```php
spl_autoload_register( autoload_function: 'classAutoLoader');

function redirect($location){
    header( string: "Location: {$location}");
}
```

Message function

Net zoals bij het proceduraal programmeren schrijven we ook hier een methode om een bericht op het scherm weer te geven. Hier schrijven we momenteel enkel de code maar maken er momenteel nog geen gebruik van. Voeg de onderstaande functions toe in Session.php.

De **check_message methode** bekijkt of er reeds een vorig bericht als sessie in het geheugen is geladen. Indien dit het geval is gaan we die **legen** d.m.v. de **unset functie.**

```php
public function message($msg=""){
    if(!empty($msg)){
        $_SESSION['message'] = $msg;
    }
    else{
        return $this->message;
    }
}

private function check_message(){
    if(isset($_SESSION['message'])){
        $this->message = $_SESSION['message'];
        unset($_SESSION['message']);
    }else{
        $this->message = "";
    }
}
```

De check_message functie dient dus eigenlijk geladen te worden telkens er een nieuwe sessie door de gebruiker wordt gestart! Daarom plaatsen we deze in de __construct van de session class.

```php
/*automatisch starten van een session*/
function __construct(){
    session_start();
    $this->check_the_login();
    $this->check_message();
}
```

1.2.4. ADMIN/LOGIN.PHP

In dit gedeelte gaan we de login pagina creëeren voor de backend.
We gebruiken terug het init.php bestand om onze connectie en classes te includen in deze pagina.

We **testen** eerste of er een **sessie actief** is. Bij een positief antwoord van de functie **is_signed_in(),** redirecten we naar de index.php van het admin gedeelte, m.a.w. de backend van de site.

Wanneer er **geen sessie actief** is wordt het onderstaande uitgevoerd.
De login.php pagina zal ook uit een gedeelte HTML bestaan waarin een form tag is opgemaakt met de velden username en paswoord samen met de login button. De login button geven we de naam **'submit'**.
Indien er op de knop wordt gedrukt om een user in te loggen dan schiet de **$_POST** variabele in gang en zullen we een trim uitvoeren om alle overbodige tekens zoals spaties uit te filteren. Hierdoor creëeren we de variabelen **$username** en **$password**.

De variabele **$user_found** krijgt de user die uit de User class wordt gehaald d.m.v. de **verify_user** methode die we nog dienen te schrijven.
Wanneer de user gevonden is dan kan er ingelogd worden naar de index.php pagina van het admin gedeelte. In het andere geval zal de gebruiker een foutmelding krijgen.

```php
<?php require_once( "includes/header.php" );?>

<?php
    if($session->is_signed_in()){
        redirect( location: "index.php");
    }

    if(isset($_POST['submit'])){
        $username = trim($_POST['username']);
        $password = trim($_POST['password']);

        //Methode check database als de user bestaat!
        $user_found = User::verify_user($username, $password);

        if($user_found){
            $session->login($user_found);
            redirect( location: "index.php");
        } else {
            $the_message = "Your password or username FAILED!";
        }
    } else {
        $username = "";
        $password = "";

    }

?>
<div class="col-md-4 col-md-offset-5">
    <form action="" method="post">
        <div class=form-grooup>
            <label for="username">Username</label>
            <input type="text" class="form-control" name="username">
        </div>
        <div class="form-group">
            <label for="password">Password</label>
            <input type="password" class="form-control" name="password">
        </div>
        <div class="form-group">
            <input type="submit" name="submit" value="Submit" class="btn btn-primary">
        </div>
    </form>
</div>
```

We dienen nu nog enkel de verify_user methode te schrijven in de user class. Open admin/includes/User.php en pas als volgt aan:

```php
public static function verify_user(){
    global $database;
    $username = $database->escape_string($username);
    $password = $database->escape_string($password);

    $sql = "SELECT * FROM users WHERE ";
    $sql .= "username = '{$username}' ";
    $sql .= "AND password = '{$password}' ";
    $sql .= "LIMIT 1";

    $the_result_array = self::find_this_query($sql);
    return !empty($the_result_array) ? array_shift($the_result_array): false;
}

public static function instantantion($found_user){
```

In admin/includes/header.php zullen we ook zorgen voor outputbuffering.

```php
<?php include("init.php"); ?>
<?php ob_start(); ?>
<!DOCTYPE html>
<html lang="en">
```

1.2.5. ADMIN/LOGOUT.PHP

Zorg er eerst voor dat je de link aanpast in content-top.php.

```
<div class="dropdown-divider"></div>
<a class="dropdown-item" href="logout.php" data-toggle="modal" data-target="#logoutModal">
    <i class="fas fa-sign-out-alt fa-sm fa-fw mr-2 text-gray-400"></i>
    Logout
</a>
```

Het modal scherm die wordt geactiveerd hierboven door het attribuut data-target kunt je terugvinden in admin/includes/footer.php:

```
<!-- Logout Modal-->
<div class="modal fade" id="logoutModal" 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">Ready to Leave?</h5>
                <button class="close" type="button" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">×</span>
                </button>
            </div>
            <div class="modal-body">Select "Logout" below if you are ready to end your current session.</div>
            <div class="modal-footer">
                <button class="btn btn-secondary" type="button" data-dismiss="modal">Cancel</button>
                <a class="btn btn-primary" href="logout.php">Logout</a>
            </div>
        </div>
    </div>
</div>
```

In de pagina admin/logout.php pas je als volgt aan:
je roept de logout() methode op die we in de session class hebben geschreven. Daarna redirect je naar de login pagina.

```
<?php require_once( "includes/header.php" );?>
<?php
$session->logout();
redirect( location: 'login.php');
```

1.2.6. MESSAGE EN AUTOINVULLING

Wanneer we naar de login pagina surfen en we geven een verkeerde username en paswoord ingeven dan zullen we een bericht weergeven voor de gebruiker.
Dit doen we door **echo $the_message** aan de pagina toe te voegen.

Declareer eerst de variabele, zodat je geen undefined error krijgt:

```
/*controle of er reeds is ingelogd**/
$the_message = '';
if($session->is_signed_in()){
    redirect( location: "index.php");
```

We willen vermijden dat gebruiker al zijn velden telkens weer opnieuw dient in te vullen. Wanneer hij een fout heeft gemaakt in 1 veld dan dien hij enkel dit veld aan te passen. Daarom gaan we alle reeds ingegeven velden terug in het formulier oproepen ter aanpassing. Dit doen we met **echo htmlentities($password).**

```
<h2 class="bg-danger"><?php echo $the_message; ?></h2>
<form action="" method="post">
    <div class=form-group>
        <label for="username">Username</label>
        <input type="text" class="form-control" name="username" value="<?php echo htmlentities($username);?>">
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <input type="password" class="form-control" name="password" value="<?php echo htmlentities($password);?>">
```

2. CRUD

CRUD = CREATE READ UPDATE DELETE.
Het Crud principe gaat over de basisfunctionaliteiten die we nodig hebben om te communiceren met onze database vertrekkende vanuit onze code. Ieder class of object die we hebben geschreven zal hiervan gebruik moeten maken.

2.1. CREATE

De create methode staat equivalent met de INSERT INTO query die we gaan gebruiken om bestanden toe te voegen in onze database.
We beginnen met de **user class** als voorbeeld.
Maak een nieuwe functie aan **create().**
Het is de bedoeling dat we een user automatisch gaan toevoegen aan onze database.

```php
public function create(){
    global $database;

    $sql = "INSERT INTO users (username, password, first_name, last_name)";
    $sql .= " VALUES ('";
    $sql .= $database->escape_string($this->username) . "', '";
    $sql .= $database->escape_string($this->password) . "', '";
    $sql .= $database->escape_string($this->first_name) . "', '";
    $sql .= $database->escape_string($this->last_name) . "')";

    if($database->query($sql)){
        $this->id = $database->the_insert_id();
        return true;
    }else{
        return false;
    }

    $database->query($sql);

}
```

Open Database.php en pas als volgt aan:
de **mysqli_insert_id** functie haalt de laatst uitgevoerde id op in een tabel die een primary key met autoincrement heeft! D.w.z. dat we met deze functie weten wat het laatst toegevoegde recordnummer was.

```php
public function the_insert_id(){
    return mysqli_insert_id($this->connection);
}
```

Open content.php en pas volgende aan:

```php
$user = new User();
$user->username = "Sam";
$user->password = "123";
$user->first_name = "Sam";
$user->last_name = "Decursist";

$user->create();
```

Test dit nu door te surfen naar admin/includes/index.php
Wanneer je niet ingelogd bent dien je dit eerst te doen om de content.php pagina te laten uitvoeren.
Controleer nu je database, de user werd toegevoegd!

2.2. UPDATE

De update methode staat equivalent met de UPDATE query die we gaan gebruiken om bestanden te wijzigen in onze database.
We beginnen met de **user class** als voorbeeld.
Maak een nieuwe functie aan **update ().**
Het is de bedoeling dat we een user automatisch gaan wijzigen in onze database.
Voeg onderstaande functie toe aan je User.php class:

```php
public function update(){
    global $database;
    $sql = "UPDATE users SET ";
    $sql .= "username= '" . $database->escape_string($this->username) . "', ";
    $sql .= "password = '" . $database->escape_string($this->password) . "', ";
    $sql .= "first_name = '" . $database->escape_string($this->first_name) . "', ";
    $sql .= "last_name = '" . $database->escape_string($this->last_name) . "' ";
    $sql .= " WHERE id= " . $database->escape_string($this->id);

    $database->query($sql);
    return (mysqli_affected_rows($database->connection) == 1) ? true : false;
}
```

De functie **mysqli_affected_rows** geeft het aantal rijen terug die door de query zullen aangepast worden.

Open content.php en pas volgende aan:

```php
$user = User::find_user_by_id( user_id: 1);
$user->last_name = "WILLIAMS";

$user->update();
```

Test dit nu door te surfen naar admin/includes/index.php
Wanneer je niet ingelogd bent dien je dit eerst te doen om de content.php pagina te laten uitvoeren.
Controleer nu je database, de user werd gewijzigd!

Resultaat:

+ Opties

	id	username	password	first_name	last_name
Wijzigen Kopiëren Verwijderen	1	tom	123456	tom	WILLIAMS

2.3. DELETE

De delete methode staat equivalent met de DELETE query die we gaan gebruiken om bestanden te verwijderen in onze database.
We beginnen met de **user class** als voorbeeld.
Maak een nieuwe functie aan **update ().**
Het is de bedoeling dat we een user automatisch gaan verwijderen in onze database.
Voeg onderstaande functie toe aan je User.php class:

```php
public function delete(){
    global $database;

    $sql = "DELETE FROM users ";
    $sql .= " WHERE id= " . $database->escape_string($this->id);
    $sql .= " LIMIT 1";

    $database->query($sql);
    return (mysqli_affected_rows($database->connection) == 1) ? true : false;

}
```

Open content.php en pas volgende aan:

```php
$user = User::find_user_by_id( user_id: 3);
$user->delete();
```

Test dit nu door te surfen naar admin/includes/index.php
Wanneer je niet ingelogd bent dien je dit eerst te doen om de content.php pagina te laten uitvoeren.
Controleer nu je database, de user werd verwijderd!

3. ABSTRACTION VAN CODE

Wat is abstraction van code nu?
Eigenlijk nemen we een groot stuk code, die we verkleinen maar bijgevolg de efficiëntie ervan vergroten in onze code.

Een voorbeeld van een methode die we dienen te vereenvoudigen is de create() methode en de update methode die we daarnet al hebben geschreven in de class User.php:
Op dit ogenblik is de create methode alleen maar in staat om telkens weer dezelfde regel te schrijven zoals je hieronder kan zien, hetzelfde zou gelden voor de update methode ook!

```php
public function create(){
    global $database;

    $sql = "INSERT INTO users (username, password, first_name, last_name)";
    $sql .= " VALUES ('";
    $sql .= $database->escape_string($this->username) . "', '";
    $sql .= $database->escape_string($this->password) . "', '";
    $sql .= $database->escape_string($this->first_name) . "', '";
    $sql .= $database->escape_string($this->last_name) . "')";

    if($database->query($sql)){
        $this->id = $database->the_insert_id();
        return true;
    }else{
        return false;
    }

    $database->query($sql);

}
```

Dit gaan we nu vereenvoudigen:
We gaan een methode creëeren die detecteert of er een user is. Wanneer deze er IS dan gaan we deze wijzigen = **updaten**. Wanneer deze er NIET is dan gaan we de user aanmaken = **createn**.

```php
public function save(){
    return isset($this->id) ? $this->update() : $this->create();
}

public function create(){
```

Open content.php en pas volgende aan:
Wanneer we een user willen **wijzigen**:

```php
$user = User::find_user_by_id( user_id: 1);
$user->username= "TEST";
$user->save();
```

Wanneer we een **nieuwe user** willen aanmaken:

```php
$user = new User();
$user->username= "TEST";
$user->save();
```

Test dit nu door te surfen naar admin/includes/index.php
Wanneer je niet ingelogd bent dien je dit eerst te doen om de content.php pagina te laten uitvoeren.
Controleer nu je database, de user werd verwijderd!

3.1. CRUD abstractie universele class

Momenteel hebben we dus een CRUD functionaliteit voor de user class. Het zou handig zijn moesten we dezelfde code voor alle classes kunnen gebruiken. Stel je eens voor dat je tabel producten, mensen, foto's, ... heeft die elk een class zijn in je project. Dezelfde code kunnen gebruiken zou dus een ongelooflijke meerwaarde zijn voor iedere nieuwe class die je creëert.

In de User.php class voeg je onderstaande protected static variabele toe.

```php
<?php
class User{
    protected static $db_table = "users";
```

```php
public function create(){
    global $database;

    $sql = "INSERT INTO " .self::$db_table . " (username, password, first_name, last_name)";
```

```php
public function update(){
    global $database;

    $sql = "UPDATE " .self::$db_table . " SET ";
```

```php
public function delete(){
    global $database;

    $sql = "DELETE FROM " .self::$db_table . " ";
```

Verander de function **find_all_users** naar **find_all**. We willen dit niet alleen voor users gebruiken.

```php
public static function find_all(){
    return self::find_this_query( sql: "SELECT * FROM " . self::$db_table . " ");

}
```

Verander de function **find_users_by_id** naar **find_by_id**. We willen dit niet alleen voor users gebruiken.

```php
public static function find_by_id($id){
    $result = self::find_this_query( sql: "SELECT * FROM " . self::$db_table . " WHERE id=$id LIMIT 1");
    return !empty($result) ? array_shift( &array: $result) : false;
}
```

Als laatste voeren we dit ook nog uit bij de functie verify_user:

```php
public static function verify_user($username, $password){
    global $database;
    $username = $database->escape_string($username);
    $password = $database->escape_string($password);

    $sql = "SELECT * FROM " . self::$db_table . " WHERE ";
```

Test dit maar uit in content.php indien je wenst! Voeg gebruikers toe, wijzig ze of delete ze. Probeer maar uit!

3.2. CRUD abstractie properties

Net zoals de tabelnaam kunnen we ook de properties op dezelfde manier benaderen.

Wanneer we kijken naar de create method en de update method van de User.php class dan staan alle properties er nog hardcoded ingegeven. Dit dienen we variabel te maken.

We dienen dus een associatieve array te maken om te loopen tussen de velden en zijn waarden.

Open de class User.php en pas als volgt aan:
we gaan een functie properties() schrijven die alle properties van de class zal inlezen.

```
protected function properties(){
    return get_object_vars($this);
}

public function save(){
```

Deze properties heb ik nu nodig binnen CRUD.
Pas het volgende aan in de **function create():**
we gebruiken de juist geschreven **properties() functie** om alle variabelen van de class in een variabele $properties te steken.
Deze properties willen we nu gebruiken in het insert statement. Aangezien dat ieder veld in het statement apart wordt ingevoerd moeten we de **$properties** gaan **imploden (in stukken verdelen)**.
Bij iedere komma in de associatieve array $properties, gaan we iedere aparte key =(veldnamen) er uit halen. Dit doen we door de functie **array_keys($properties)** te gebruiken.

```
public function create(){
    global $database;
    $properties = $this->properties();

    $sql = "INSERT INTO " .self::$db_table . "(" . implode( glue: ",", array_keys($properties)) .")";
```

Het volgende die we dienen uit te voeren zijn de values van iedere key uit de associatieve array te trekken.

```
public function create(){
    global $database;
    $properties = $this->properties();

    $sql = "INSERT INTO " .self::$db_table . "(" . implode( glue: ",", array_keys($properties)) .")";
    $sql .= " VALUES ('" . implode( glue: "','", array_values($properties))   . "')";
```

PROBLEEM! Wanneer we bovenstaande $sql zouden uitvoeren dan zouden we een ERROR krijgen omdat er één veld teveel met de properties is meegekomen, nl de tabelnaam zelf.

```
protected static $db_table = "users";
public $id;
public $username;
public $password;
public $first_name;
public $last_name;
```

Dit lossen we als volgt op:
we declareren alle variabelen in een aparte variabele **$db_table_fields**

```
class User{
    protected static $db_table = "users";
    protected static $db_table_fields = array('username', 'password','first_name', 'last_name');
```

In function properties() is de **get_object_vars($this)** nu overbodig aangezien deze functie alle object variabelen uit user class verzamelde.
In een gewone array $properties gaan we nu alleen de velden van de tabel users laden.

We doorlopen de array variabele $db_table_fields veld per veld. Met **property_exists** controleren we of de 2 parameters bestaan. De eerste is de class zelf, nl $this (User). De tweede is de property van de class. Indien beide aanwezig zijn in de class dan creëeren we een dynamische variabele **$properties[$db_field] die de veldnaam (key) bevat**.

```php
protected function properties(){
    //return get_object_vars($this);
    $properties = array();
    foreach(self::$db_table_fields as $db_field){
        if(property_exists($this,$db_field)){
            $properties[$db_field] = $this->$db_field;
        }
    }
    return $properties;
}
```

Probeer maar uit! Creëer een nieuwe user in admin/includes/content.php!

Pas het volgende aan in de **function update():**
We gaan alle properties van de class in een **associatieve array** plaatsen, nl. **$properties_assoc**. Deze array wordt gevuld met het veld en de value zoals de schrijfwijze van het sql statement dit verwacht, nl. "{$key}='{$value}'"

```php
public function update(){
    global $database;
    $properties = $this->properties();
    $properties_assoc = array();

    foreach($properties as $key => $value){
        $properties_assoc[] = "{$key}='{$value}'";
    }

    $sql = "UPDATE " .self::$db_table . " SET ";
    $sql .= implode(glue: ", ", $properties_assoc);
    $sql .= " WHERE id= " . $database->escape_string($this->id);

    $database->query($sql);
    return (mysqli_affected_rows($database->connection) == 1) ? true : false;

}
```

Het enige wat we nog vergeten zijn is de escape_string voor de values die via een insert (create) in de database worden ingevoegd.
Pas in User.php als volgt aan:
we schrijven een nieuwe methode **clean_properties().**
We nemen de vorige **methode** :$this->**properties** en doorlopen zowel de key als de value. De value gaan we opschonen met **escape_string(value)** en steken we telkens in de array **$clean_properties**.

```php
protected function clean_properties(){
    global $database;
    $clean_properties = array();
    foreach($this->properties() as $key => $value){
        $clean_properties[$key] = $database->escape_string($value);
    }
    return $clean_properties;
}
```

Het enige dat we nu nog dienen te veranderen zijn de create en de update function:

```php
public function create(){
    global $database;
    $properties = $this->clean_properties();
```

```php
public function update(){
    global $database;
    $properties = $this->clean_properties();
```

4. PARENT CLASS

Tot nu toe stonden al onze methodes (functies) in de User.php class. Heel wat methodes kunnen dus nu al dienen voor andere toekomstige classes, zoals de photo class er één zal zijn. We dienen dus alle gemeenschappelijke methodes in een basis class te steken. Zo een class noemen de **parent** of de base class.

ga naar **admin/includes** en voeg een nieuw bestand toe **Db_object.php, dit is de parent class!**
We gaan nu alle gemeenschappelijke classes uit User.php **knippen** en deze in de parent class db_object **plakken**.

De methodes die je dient te knippen en plakken zijn:
find_all, find_by_id, find_this_query, instantantie, has_the_attribute, save, create, update, delete, properties en clean_properties.

De class User.php is dus zijn methodes kwijt, nu dienen we gebruik te maken van overerving door in de class User.php de parent class Db_object.php aan te spreken. Overerving dient te gebeuren door de huidige classe uit te breiden met de parent class (= **extends**)

```
<?php
class User extends Db_object {
```

Daarnaast dienen we onze init.php bestand ook te voorzien van een include van onze nieuwe class.

```
require_once("functions.php");
require_once("new_config.php");
require_once("database.php");
require_once("db_object.php");
require_once("user.php");
require_once("session.php");
```

PROBLEEM! Wanneer we gebruik maken van overerving dan kan je GEEN gebruik maken van **STATIC methods (self)**.
OPLOSSING: we dienen dit dus voor alle methodes in de **parent class Db_object.php aan te passen**.
Wijzig hiervoor alle methodes in Db_object.php waar het keyword **self** staat naar **static.**
LET OP: enkel in de instantantie method mogen we dit niet doen, omdat deze methode verwijst naar de functie zelf.
De OPLOSSING hiervoor noemen STATIC LATE BINDING. Pas als volgt aan:

```php
public static function instantie($result){
    $calling_class = get_called_class(); //late static binding
    $the_object = new $calling_class;

    foreach($result as $the_attribute => $value){
        if($the_object->has_the_attribute($the_attribute)){
            $the_object->$the_attribute = $value;
        }
    }
    return $the_object;
}
```

Surf nu naar admin/index.php en voer volgende code uit! Dit zou nu alle users op het scherm moeten weergeven.
admin/includes/admin_content

```php
$users = User::find_all();
foreach ($users as $user){
    echo $user->username;
}
```

5. PHOTO CLASS

5.1. TABEL: photos

Open je database en voeg de volgende tabel toe: photos!

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1	**photo_id**	int(11)			No	*None*		AUTO_INCREMENT
☐	2	**title**	varchar(255)	utf8_general_ci		No	*None*		
☐	3	**description**	text	utf8_general_ci		No	*None*		
☐	4	**filename**	varchar(255)	utf8_general_ci		No	*None*		
☐	5	**type**	varchar(255)	utf8_general_ci		No	*None*		
☐	6	**size**	int(11)			No	*None*		

Voeg ook volgende data toe:

		photo_id	title	description	filename	type	size
☐	Edit Copy Delete	1	Photo Belgium	Lorem ipsum dolor sit amet, consec...	image.jpg	jpg	35

5.2. PHOTOS: CLASS TOEVOEGEN

Voeg een nieuw bestand toe in admin/includes/**Photo.php**. Ook hier **erven** we van van de base class **db_Object**.

```
class Photo extends  Db_object
{
    protected static $db_table = "photos";
    protected static $db_table_fields = array('title', 'description', 'filename', 'type', 'size');
    public $photo_id;
    public $title;
    public $description;
    public $filename;
    public $type;
    public $size;
```

Zorg ervoor dat je dit bestand onmiddellijk gaat includen in admin/includes/init.php:

require_once("Photo.php")

Test dit nu even uit in admin/includes/**content.php**.

```
$photos= Photo::find_all();
foreach($photos as $photo){
    echo $photo->title . "<br>";
}
```

Probeer maar uit surf naar admin/includes/index.php. De titel van de eerste foto verschijnt op het scherm.

Probeer als oefening nu in content.php met CODE een nieuwe foto toe te voegen in de tabel photos van de database. De eigenlijke foto wordt niet opgeladen in een database (NOT DONE! Dit om vertraging te vermijden tijdens het inlezen), maar wel de link naar de locatie waar de foto werd opgeladen

```
$photo = new Photo();
$photo->title = "Sam";
$photo->description = "Lorem ipsum";
$photo->size = 15;

$photo->create();
```

5.3. PHOTOS: UPLOAD PATH EN PROPERTIES

Pas Photo.php als volgt aan:we voegen locatie en properties van de op te laden images toe.

```
public $tmp_path;
public $upload_directory = 'img';
public $errors = array();
public $upload_errors_array = array(
    UPLOAD_ERR_OK => "There is no error",
    UPLOAD_ERR_INI_SIZE => "The uploaded file exceeds the upload max_filesize from php.ini",
    UPLOAD_ERR_FORM_SIZE => "The upload file exceeds MAX_FILE_SIZE in php.ini voor html form",
    UPLOAD_ERR_NO_FILE => "No file uploaded",
    UPLOAD_ERR_PARTIAL => "The file was partially uploaded",
    UPLOAD_ERR_NO_TMP_DIR => "Missing a temporary folder",
    UPLOAD_ERR_CANT_WRITE => "Failed to write do disk",
    UPLOAD_ERR_EXTENSION => "A php extension stopped your upload"
);
```

Hierna gaan we de constanten declareren om de **site root** te bepalen, de **locatie** van de **includes** map en de **locatie** van de **images** map. Pas **init.php** als volgt aan:

```
<?php
    defined( name: 'DS') ? null : define('DS', DIRECTORY_SEPARATOR);
    define('SITE_ROOT', DS . 'wamp64' . DS . 'www' . DS . 'blog');
    defined( name: 'INCLUDES_PATH') ? null : define('INCLUDES_PATH', SITE_ROOT.DS.'admin'.DS.'includes');
    defined( name: 'IMAGES_PATH') ? null : define('IMAGES_PATH', SITE_ROOT.DS.'admin'.DS.'img');
```

OPGELET! De site root zal er misschien anders uitzien dan deze op uw pc. Zoek het volledige path op je pc die zal leiden tot de root van je blog directory en pas de 2de lijn hierboven aan!

Test nu uit of de constanten werken in admin/includes/content.php. Je zou de pad(en) moeten zien verschijnen wanneer je surft naar admin/includes/index.php

```
echo INCLUDES_PATH;
```

Maak gelijk ook de img **map** aan onmiddellijk onder de root van de admin map wanneer deze nog niet zou bestaan! Wanneer je met de rechtermuisknop in je verkenner/finder klikt dan zorg je ervoor dat je via de properties die map schrijfbaar maakt.

5.4. PHOTOS: METHODES

In de Photo.php class schrijven we eerst de code die onze errors dient op te vangen:

```php
//Hier dragen we de super global variable $_FILES['uploaded_file] over als een argument.
public function set_file($file){
    if(empty($file) || !$file || !is_array($file)) {
        $this->errors[]= "No file uploaded!";
        return false;
    }elseif($file['error'] != 0){
        $this->errors[] = $this->upload_errors_array[$file['error']];
        return false;
    }else{
        $this->filename = basename($file['name']);
        $this->tmp_path = $file['tmp_name'];
        $this->type = $file['type'];
        $this->size = $file['size'];
    }

}
```

Daarna schrijven we de save() method om de foto's op te laden naar de database.

```php
public function save(){
    if($this->photo_id) {
        $this->update();
    }else{
        if(!empty($this->errors))
        {
            return false;
        }
        if(empty($this->filename) || empty($this->temp_path)){
            $this->errors[] = "File not available";
            return false;
        }
        $target_path = SITE_ROOT . DS . "admin" . DS . $this->upload_directory . DS . $this->filename;

        if(file_exists($target_path)){
            $this->errors[] = "File {$this->filename} exists!";
            return false;
        }
        if(move_uploaded_file($this->tmp_path, $target_path)){
            if($this->create()){
                unset($this->tmp_path);
                return true;
            }
        }
        else{
            $this->errors[]= "this folder has no write rights!";
            return false;
        }
    }
}
```

5.5. PAGINA: UPLOAD

Om bovenstaande code beter te begrijpen creëeren we eerst ons upload.php bestand. Dit kun je reeds terugvinden onder de admin folder. Pas de pagina als volgt aan:

```php
<?php include("includes/header.php"); ?>
<?php include("includes/sidebar.php"); ?>
<?php include("includes/content-top.php"); ?>
<?php
if(!$session->is_signed_in()){
redirect( location: 'login.php');
}

$message = "";
if(isset($_POST['submit'])) {
$photo = new Photo();
$photo->title = $_POST['title'];
$photo->set_file($_FILES['file']);

if($photo->save()) {
$message = "Photo uploaded sucessfully";
} else {
$message = join( glue: "<br>", $photo->errors);
}
}

?>

<div class="container-fluid">
    <div class="row">
        <div class="col-12">
            <h1 class="page-header">
                UPLOAD
            </h1>
            <form action="upload.php" method="post" enctype="multipart/form-data">
                <div class="form-group">
                    <label for="title">title</label>
                    <input type="text" name="title" class="form-control">
                </div>
                <div class="form-group">
                    <input type="file" name="file" class="form-control">
                </div>
                <input type="submit" name="submit" value="submit" class="btn btn-primary">
            </form>
        </div>
    </div>
</div>
<?php include("includes/footer.php"); ?>
```

5.6. PAGINA: PHOTOS

5.6.1. PAGINA: PHOTOS WEERGEVEN

We wensen een lijst weer te geven van alle foto's zoals je hieronder kan zien.
Momenteel kunnen er 1 of meerdere foto's in de tabel photos te zien zijn.
In mijn voorbeeld zit er 1 foto in de tabel.



Open admin/photos.php en pas als volgt aan:
Enkel een **ingelogde gebruiker** mag deze pagina te zien krijgen. Hiervoor gebruiken we terug de **is_signed_in()** functie.
Daarna gaan we op deze pagina alle foto's weergeven die in de tabel photos zitten. We gebruiken hiervoor de functie **find_all()** die in de **base class** Db_object.php zit en waarvan **Photos** die **overerft**.

Het volledige object met zijn values zit nu in $photos.
We passen de pagina aan door alle foto's in een tabel weer te geven. Vervolgens lussen we door onze object variabele $photos om de data weer te geven in rijen.

```php
<?php
    include("includes/header.php");
    if(!$session->is_signed_in()){
        redirect( location: 'login.php');
    }
    $photos = Photo::find_all();
?>
<?php include("includes/sidebar.php"); ?>
<?php include("includes/content-top.php"); ?>
<div class="container-fluid">
    <div class="row">
        <div class="col-12">
            <h2>PHOTOS</h2>
            <table class="table table-header">
                <thead>
                <tr>
                    <th>Photo</th>
                    <th>Id</th>
                    <th>Title</th>
                    <th>File Name</th>
                    <th>Size</th>

                </tr>
                </thead>
                <tbody>
                <?php
                foreach($photos as $photo): ?>
                    <tr>

                        <td><img src="http://placehold.it/62x62" height="62" width="62" alt=""; ?></td>
                        <td><?php echo $photo->photo_id; ?></td>
                        <td><?php echo $photo->title; ?></td>
                        <td><?php echo $photo->filename; ?></td>
                        <td><?php echo $photo->size; ?></td>
                    </tr>
                <?php endforeach; ?>
                </tbody>
            </table>
        </div>

    </div>
</div>
<?php include("includes/footer.php"); ?>
```

Dit zou nu reeds moeten werken. Probeer maar uit! Surf naar admin/photos.php.

Vervolgens dienen we ons image path nog dynamisch te maken.
Open hiervoor je class **Photo.php** en pas aan als volgt:
de functie picture_path creëert de locatie waar onze bestanden zijn opgeladen samen met de bestandsnaam.

```php
public function picture_path() {

    return $this->upload_directory.DS.$this->filename;
}
```

Nu we deze locatie hebben kunnen we de statische html tag aanpassen waar de images in de pagina photos.php worden weergegeven.

```php
<?php
foreach($photos as $photo): ?>
    <tr>

        <td><img  src="<?php echo $photo->picture_path(); ?>" height="62"
                  width="62"
                  alt=""></td>
```

Wanneer je nu deze pagina opnieuw zou openen dan worden de foto's uit je images map ook weergegeven in de tabel.

Voeg de foto's toe aan de img map. In mijn voorbeeld is dit belgium.jpg.
Probeer maar uit!



5.6.2. PAGINA: PHOTOS VERWIJDEREN

Open admin/photos.php en pas aan als volgt:
we voegen een link toe aan iedere rij die ervoor zal zorgen dat de photo verwijderd wordt. Door te klikken op de link dienen we het id van deze foto mee te geven als parameter naar de delete_Photo.php pagina. Deze pagina zal ervoor zorgen dat we de foto effectief kunnen verwijderen.

```
<div class="container-fluid">
    <div class="row">
        <div class="col-12">
            <h2>PHOTOS</h2>
            <table class="table table-header">
                <thead>
                <tr>
                    <th>Photo</th>
                    <th>Id</th>
                    <th>Title</th>
                    <th>File Name</th>
                    <th>Size</th>
                    <th>Delete?</th>

                </tr>
                </thead>
                <tbody>
                <?php
                foreach($photos as $photo): ?>
                    <tr>

                        <td><img  src="<?php echo $photo->picture_path(); ?>" height="40"
                                  width="40"
                                  alt=""></td>
                        <td class="d-flex align-self-stretch"><?php echo $photo->photo_id; ?></td>
                        <td><?php echo $photo->title; ?></td>
                        <td><?php echo $photo->filename; ?></td>
                        <td><?php echo $photo->size; ?></td>
                        <td><a class="btn btn-danger rounded-0" href="delete_photo.php?id=<?php echo
                            $photo->photo_id; ?>"><i class="far fa-trash-alt"></i></a></td>

                    </tr>
                <?php endforeach; ?>
                </tbody>
```

De **delete_Photo.php pagina bestaat nog nie**t. Maak deze aan onder admin/delete_Photo.php en kopieer de inhoud van admin/index.php er in.
Pas vervolgens aan:
we controleren terug of een gebruiker is ingelogd. Indien niet redericten we naar de login.php pagina. Daarna kijken we na in de url of de **id** parameter door de pagina photos.php is meegegeven. Wanneer deze leeg is verwijzen we onmiddellijk terug naar de photos.php pagina. In het andere geval gebruiken we de class Photo die de methode find_by_id uit de base class db_Object overerft om het volledige object uit de database te halen.
Als het object bestaat dan gaan we deze via de object variabele $photo en zijn methode **delete_photo()** verwijderen. Let op: deze methode dienen we nog te schrijven.

```php
<?php include("includes/header.php"); ?>
<?php
if(!$session->is_signed_in()){
    redirect( location: "login.php");

}
if(empty($_GET['id'])){
    redirect( location: 'photos.php');
}

$photo = Photo::find_by_id($_GET['id']);
if($photo){
    $photo->delete_photo();
    redirect( location: 'photos.php');
}else{
    redirect( location: 'photos.php');
}
?>

<?php include("includes/sidebar.php"); ?>
<?php include("includes/content-top.php"); ?>
<h1>welkom delete pagina</h1>
<?php include("includes/footer.php"); ?>
```

ENKELE CODE AANPASSINGEN

Het maakt niet uit of we nu een id ophalen bij een user of foto, de methode blijft dezelfde. Pas hiervoor de onderstaande methode aan als volgt. (Db_object.php)

```php
public static function find_by_id($id){
    global $database;
    $the_result_array = static::find_this_query( sql: "SELECT * FROM " . static::$db_table . " WHERE id= $id LIMIT 1");
    return !empty($the_result_array) ? array_shift($the_result_array): false;
}
```

Open de class Photo.php en pas alle **photo_id** ook aan naar **id**.

Vergeet daarnaast ook niet in de tabel photos van je database het veld **photo_id** ook niet te wijzigen naar **id**.

	#	Naam	Type	Collatie	Attributer
☐	1	id 🔑	int(11)		
☐	2	title	varchar(255)	latin1_swedish_ci	
☐	3	description	text	latin1_swedish_ci	
☐	4	filename	varchar(255)	latin1_swedish_ci	
☐	5	type	varchar(255)	latin1_swedish_ci	
☐	6	size	int(11)		

Ook op de pagina photos.php dien je deze te wijzigen.

```php
<?php
foreach($photos as $photo): ?>
    <tr>

        <td><img  src="<?php echo $photo->picture_path(); ?>" height="40"
                  width="40"
                  alt=""></td>
        <td class="d-flex align-self-stretch"><?php echo $photo->id; ?></td>
        <td><?php echo $photo->title; ?></td>
        <td><?php echo $photo->filename; ?></td>
        <td><?php echo $photo->size; ?></td>
        <td><a class="btn btn-danger rounded-0" href="delete_photo.php?id=<?php echo
            $photo->id; ?>"><i class="far fa-trash-alt"></i></a></td>

    </tr>
<?php endforeach; ?>
```

Delete_photo methode

Open de class Photo.php en voeg de delete_photo() method toe:
Wanneer er op de link in photos.php wordt geklikt dan dient deze methode getriggerd te worden. We gebruiken in deze methode de **standaard delete() methode** die we **overerven** van **Db_object.php**.
Herinner je dat deze methode enkel de verwijdering in de database zal uitvoeren.
De variabele target_path bevat de locatie en het bestand van de foto die we wensen te verwijderen.
Deze variabele target_path voederen we als parameter aan de built-in functie unlink. **Unlink** verwijderd het bestand in de map zelf.

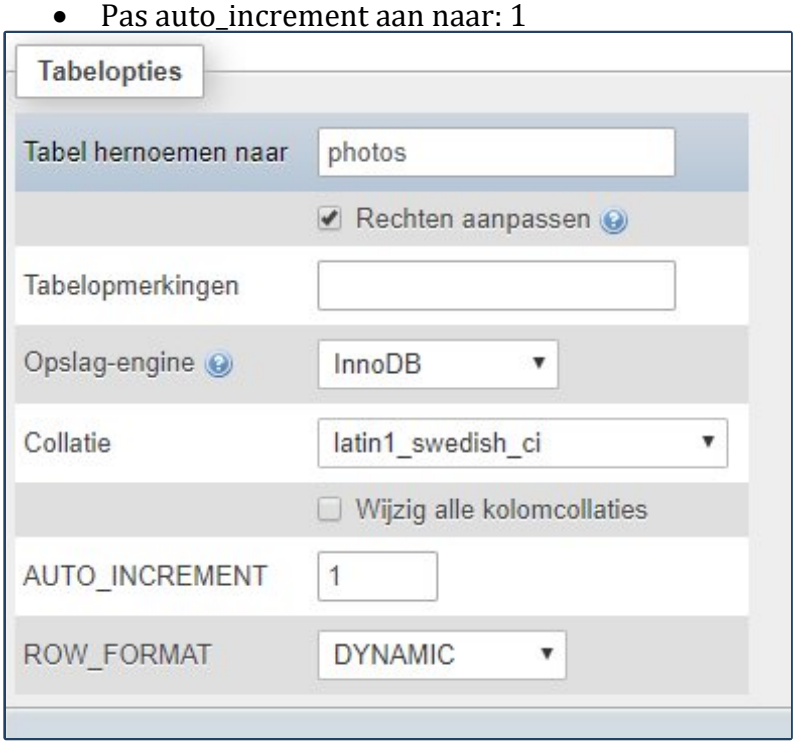
```
public function delete_photo() {
    if($this->delete()) {
        $target_path = SITE_ROOT.DS. 'admin' . DS . $this->picture_path();
        return unlink($target_path) ? true : false;
    } else {
        return false;
    }
}
```

Probeer maar uit en delete de foto.
Resultaat:



Je foto's zijn nu leeg in de tabel photos van je database. Wanneer je nu een nieuwe foto zou willen toevoegen dan zou je denken dat het id terug vanaf zal beginnen nummeren. Dit is niet zo!
Phpmyadmin zal verder nummeren vanaf id 2.
Wanneer je terug opnieuw vanaf 1 wenst te nummeren dan voer je in phpmyadmin volgende stappen uit:

- Klik op de tabel photos
- Klik op het tabblad Handelingen
- Pas auto_increment aan naar: 1



5.6.3. PAGINA: PHOTOS WIJZIGEN

Deze pagina noemen admin/edit_Photo.php. Kopieer en plak de inhoud van photos.php.

We dienen eerst ons html formulier te maken.
Zorg er wel eerst voor dat onderstaande code in commentaar staat.

```php
<?php
    //$photos = Photo::find_all();
?>
```

Verwijder het formulier van de vorige pagina en voeg dit nieuwe formulier toe:

```
</h1>
<form action="edit_photo.php" method="post">
    <div class="col-md-8">
        <div class="form-group">
            <input type="text" name="title" class="form-control">
        </div>
        <div class="form-group">
            <label for="caption">Caption</label>
            <input type="text" name="caption" class="form-control">
        </div>
        <div class="form-group">
            <label for="caption">Alternate Text</label>
            <input type="text" name="alternate_text" class="form-control">
        </div>
        <div class="form-group">
            <label for="caption">Description</label>
            <textarea class="form-control" name="description" id="" cols="30" rows="10"></textarea>
        </div>
    </div>
    <div class="col-md-4" >
        <div  class="photo-info-box">
            <div class="info-box-header">
                <h4>Save <span id="toggle" class="glyphicon glyphicon-menu-up pull-right"></span></h4>
            </div>
            <div class="inside">
                <div class="box-inner">
                    <p class="text">
                        <span class="glyphicon glyphicon-calendar"></span> Uploaded on: April 01, 2020 @ 5:26
                    </p>
                    <p class="text ">
                        Photo Id: <span class="data photo_id_box">34</span>
                    </p>
                    <p class="text">
                        Filename: <span class="data">image.jpg</span>
                    </p>
                    <p class="text">
                        File Type: <span class="data">JPG</span>
                    </p>
                    <p class="text">
                        File Size: <span class="data">3245345</span>
                    </p>
                </div>
                <div class="info-box-footer clearfix">
                    <div class="info-box-delete pull-left">
                        <a  href="delete_photo.php?id=<?php echo $photo->id; ?>" class="btn btn-danger btn-lg ">Delete</a>
                    </div>
                    <div class="info-box-update pull-right ">
                        <input type="submit" name="update" value="Update" class="btn btn-primary btn-lg ">
                    </div>
                </div>
            </div>
        </div>
    </div>
</form>
```

In het formulier kan je zien dat we 2 extra velden hebben toegevoegd nl. caption en alternate text.
Voeg deze ook toe in de database:

	#	Name	Type	Collation	Attributes	Null	Default
☐	1	id	int(11)			No	None
☐	2	title	varchar(255)	utf8_general_ci		No	None
☐	3	caption	varchar(255)	utf8_general_ci		No	None
☐	4	description	text	utf8_general_ci		No	None
☐	5	filename	varchar(255)	utf8_general_ci		No	None
☐	6	alternate_text	varchar(255)	utf8_general_ci		No	None
☐	7	type	varchar(255)	utf8_general_ci		No	None
☐	8	size	int(11)			No	None

In de class Photo.php dienen we deze velden natuurlijk ook toe te voegen.

```php
class Photo extends  Db_object
{
    protected static $db_table = "photos";
    protected static $db_table_fields = array('title', 'caption', 'description', 'filename', 'alternate_text', 'type', 'size');
    public $id;
    public $title;
    public $caption;
    public $description;
    public $filename;
    public $alternate_text;
    public $type;
    public $size;
```

De submit button die we in ons formulier hebben gebruikt heeft de naam 'update'. We zorgen er dus bovenaan onze pagina voor, dat alle velden die in het formulier gepost kunnen worden, overgedragen worden een andere pagina. In dit geval de edit_php pagina.
De edit pagina dient hiervoor voorzien te zijn van onderstaande link die zorgt voor de overdracht van gegevens.
Open photos.php en voeg de volgende wijzigingen door:

```php
<table class="table table-header">
    <thead>
    <tr>
        <th>Photo</th>
        <th>Id</th>
        <th>Title</th>
        <th>Caption</th>
        <th>File Name</th>
        <th>Alternate Text</th>
        <th>Size</th>
        <th>Wijzig?</th>
        <th>Delete?</th>

    </tr>
    </thead>
    <tbody>
    <?php
    foreach($photos as $photo): ?>
        <tr>

            <td><img  src="<?php echo $photo->picture_path(); ?>" height="40"
                      width="40"
                      alt=""></td>
            <td class="d-flex align-self-stretch"><?php echo $photo->id; ?></td>
            <td><?php echo $photo->title; ?></td>
            <td><?php echo $photo->caption; ?></td>
            <td><?php echo $photo->filename; ?></td>
            <td><?php echo $photo->alternate_text; ?></td>
            <td><?php echo $photo->size; ?></td>
            <td><a class="btn btn-danger rounded-0" href="edit_photo.php?id=<?php echo
                $photo->id; ?>"><i class="far fa-edit"></i></a></td>
            <td><a class="btn btn-danger rounded-0" href="delete_photo.php?id=<?php echo
                $photo->id; ?>"><i class="far fa-trash-alt"></i></a></td>

        </tr>
    <?php endforeach; ?>
```

Zorg er nu eerst voor dat je terug een foto in je database handmatig creëert .

Vooraleer we de update kunnen uitvoeren dienen we eerst ons formulier dus te vullen met bestaande data, m.a.w. een foto met zijn gegevens.

Open **edit_Photo.php** en pas aan als volgt:
Eerst controleren we of de parameter id die binnen komt niet leeg is. Wanneer dit leeg is, verwijzen we onmiddellijk terug naar de photos.php pagina.
In het andere geval gebruiken we de method find_by_id die een query uitvoert in de database en het resultaat in de object variabele $photo steekt.
Het updaten zal kunnen gebeuren wanneer een gebruiker in het formulier op de update button klikt. In de object variabele $photo wordt nu ieder veld gevuld met de global post variabelen die werden overgedragen uit het formulier. Op die manier kunnen we terug gebruik maken van onze object variabelen in het verder verloop van onze pagina.
We voegen hier ook de update functie toe om de wijzigingen van het formulier weg te schrijven naar de database.

```php
include("includes/header.php");
if (!$session->is_signed_in()) {
    redirect( location: "login.php");

}
if (empty($_GET['id'])) {
    redirect( location: 'photos.php');
}else{
    $photo = Photo::find_by_id($_GET['id']);
    if(isset($_POST['update'])){
        if($photo){
            $photo->title = $_POST['title'];
            $photo->caption = $_POST['caption'];
            $photo->description = $_POST['description'];
            $photo->alternate_text = $_POST['alternate_text'];
            $photo->type = $_POST['type'];
            $photo->size = $_POST['size'];
            $photo->update();
        }
    }
}
```

Nu kunnen we ons formulier vullen met de object variabelen:

```
<div class="container-fluid">
    <div class="row">
        <div class="col-12">
            <h1>welkom edit pagina</h1>

            <form action="edit_photo.php?id=<?php echo $photo->id; ?>" method="post">
                <div class="row">
                    <div class="col-12 col-md-8">
                        <div class="form-group">
                            <label for="title">Title</label>
                            <input type="text" name="title" class="form-control"

                                    value="<?php echo $photo->title; ?>">
                        </div>
                        <div class="form-group">
                            <a class="thumbnail" href="#"><img src="<?php echo $photo->picture_path(); ?>" alt=""></a>
                        </div>
                        <div class="form-group">
                            <label for="caption">Caption</label>
                            <input type="text" name="caption" class="form-control"
                                    value="<?php echo $photo->caption; ?>">
                        </div>
                        <div class="form-group">
                            <label for="description">Description</label>
                            <textarea class="form-control" name="description" id="" cols="30" rows="10"><?php echo $photo->description; ?></textarea>

                        </div>
                        <div class="form-group">
                            <label for="alternate_text">Alternate Text</label>
                            <input type="text" name="alternate_text" class="form-control"
                                    value="<?php echo $photo->alternate_text; ?>">
                        </div>
                    </div>
```

Vervolg: zie volgende pagina.

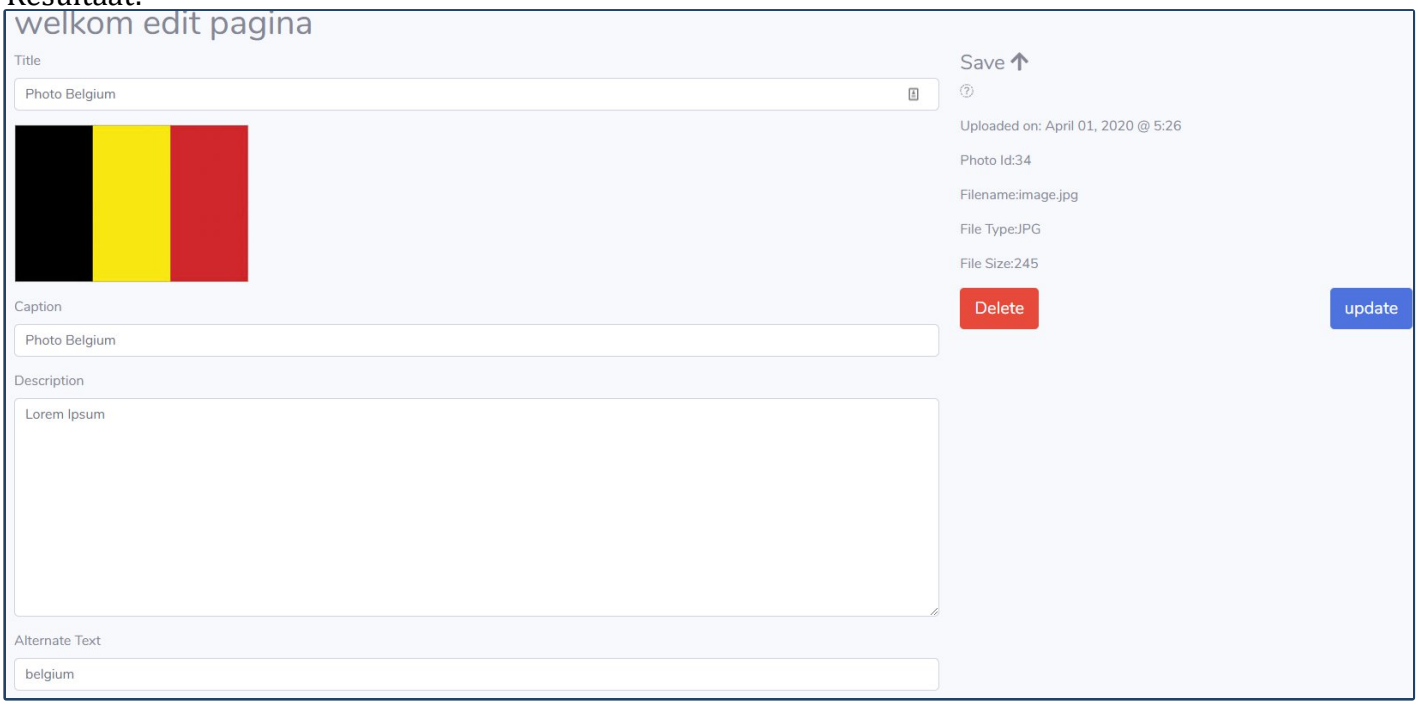
```
                    <div class="col-12 col-md-4">
                        <div class="photo-info-box">
                            <div class="info-box-header">
                                <h4>Save <span id="toggle" class="fas fa-arrow-up"></span></h4>
                            </div>
                            <div class="inside">
                                <div class="box-inner">
                                    <p class="text">
                                        <span class="fas fa-calendar">Uploaded on: April 01, 2020 @ 5:26</span>
                                    </p>
                                    <p class="text">
                                        <span class="data photo_id_box">Uploaded on: April 01, 2020 @
                                                                                5:26</span>
                                    </p>
                                    <p class="text">
                                        Photo Id:<span class="data">34</span>
                                    </p>
                                    <p class="text">
                                        Filename:<span class="data">image.jpg</span>
                                    </p>
                                    <p class="text">
                                        <label for="type">File Type:</label>
                                        <input readonly type="text" name="type" class="form-control"
                                               value="<?php echo $photo->type; ?>">

                                    </p>
                                    <p class="text">
                                        <label for="size">File Type:</label>
                                        <input readonly type="text" name="size" class="form-control"
                                               value="<?php echo $photo->size; ?>">
                                    </p>

                                </div>
                                <div class="info-box-footer">
                                    <div class="info-box-delete float-left">
                                        <a class="btn btn-danger btn-lg" href="delete_photo.php?id=<?php echo
                                        $photo->id;
                                        ?>">Delete</a>
                                    </div>
                                    <div class="info-box-update float-right">
                                        <input class="btn btn-primary btn-lg" type="submit" name="update"
                                               value="update">
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </form>
        </div>
    </div>
/div>
?php include("includes/footer.php"); ?>
```

Vergeet ook niet het id in de form zelf mee te geven. Indien je dit niet doet weten we niet welke foto aangepast dient te worden.

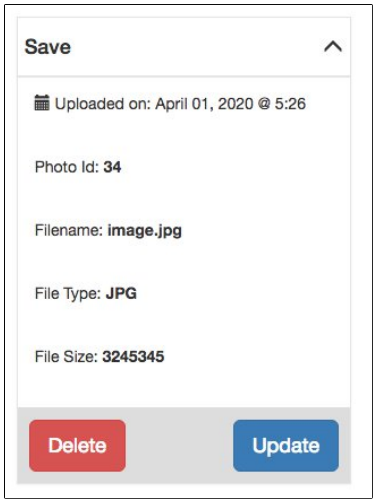
```
<h1>welkom edit pagina</h1>
<form action="edit_photo.php?id=<?php echo $photo->id; ?>" method="post">
```

Probeer maar uit, surf naar admin/photos.php! selecteer een foto en wijzig de gegevens. **Enkel de foto zelf kunnen we nu nog niet wijzigen**. Probeer maar uit nu kan je wijzigen.

Resultaat:



Stylen van de rechterkant van de edit_Photo.php pagina.
Je kan deze stylen naar eigen goeddunken. Dit kan je zelf toevoegen als oefening.
Voorbeeld:



WYSIWYG EDITOR

We zullen een text editor toevoegen ipv een standaard textarea waar we de beschrijving wegschrijven met platte tekst. Dit is een javascript editor die we dienen toe te voegen. (tinymce).

Open admin/footer.php en voeg als volgt toe:

tinymce is een tekst editor die veelvuldig wordt gebruikt in online toepassingen.

```
<script src="https://cdn.tiny.cloud/1/no-api-key/tinymce/5/tinymce.min.js" referrerpolicy="origin"></script>
<script src="js/script.js"></script>
</body>
```

Eénmaal je bovenstaande hebt ingevoegd, dienen we de bijhorende code die we via de site van tinymce hebben verkregen toe te voegen in een apart javascript bestand. Voeg hiervoor een nieuw bestand toe onder de folder js: scripts.js en plak het volgende erin:

```
tinymce.init({selector:'textarea',
    menubar: false,  // removes the menubar
});
```

Deze code zorgt ervoor dat er een tekst editor wordt getoond telkens we een textarea tag in een html form aanmaken.

Probeer maar uit! surf naar de edit_php pagina.

5.7. PAGINA: USERS

We hebben tot nu toe reeds veel methodes geschreven om de user in te loggen. Het enige wat we nog niet hebben is de afgewerkte admin pagina waar de users worden weergegeven.
Kopieer de inhoud van **photos.php** volledig over die van **users.php**. Dit is de vertrekpositie voor deze pagina.
Doe een Find/Replace voor alle **photo** naar **user**. Zorg wel dat je een hoofdletter tikt voor de class nadien: **U**ser::find_all();

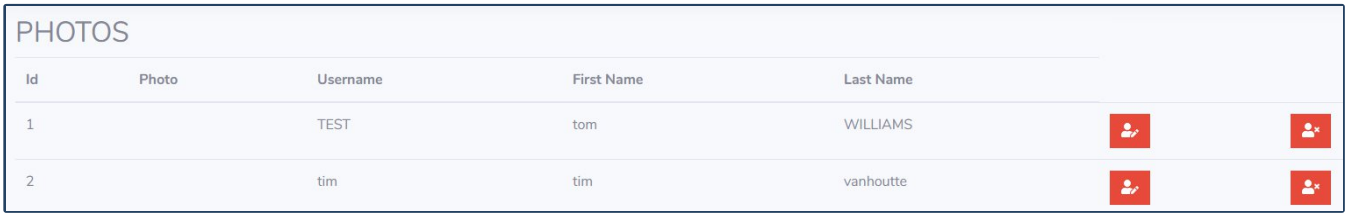
Pas nu het formulier aan in users.php als volgt:
Let op: momenteel hebben we in onze database nog geen image. We vullen een echo "test" hiervoor in plaats.

Zet eerst rond onderstaande code nog een container-fluid en een row class!

```
<?php
    include("includes/header.php");
    if(!$session->is_signed_in()){
        redirect( location: 'login.php');
    }
    $users = User::find_all();
?>
<?php include("includes/sidebar.php"); ?>
<?php include("includes/content-top.php"); ?>
<div class="container-fluid">
    <div class="row">
        <div class="col-12">
            <h2>PHOTOS</h2>
            <table class="table table-header">
                <thead>
                <tr>
                    <th>Id</th>
                    <th>Photo</th>
                    <th>Username</th>
                    <th>First Name</th>
                    <th>Last Name</th>
                </tr>
                </thead>
                <tbody>
                <?php
                foreach($users as $user): ?>
                    <tr>
                        <td><?php echo $user->id; ?></td>
                        <td><img height="40" width="40" src="<?php echo "test"; ?>" alt=""></td>

                        <td><?php echo $user->username; ?></td>
                        <td><?php echo $user->first_name; ?></td>
                        <td><?php echo $user->last_name; ?></td>
                        <td><a class="btn btn-danger rounded-0" href="edit_user.php?id=<?php echo
                            $user->id; ?>"><i class="fas fa-user-edit"></i></a></td>
                        <td><a class="btn btn-danger rounded-0" href="delete_user.php?id=<?php echo
                            $user->id; ?>"><i class="fas fa-user-times"></i></a></td>

                    </tr>
                <?php endforeach; ?>
                </tbody>
            </table>
        </div>

    </div>
</div>
<?php include("includes/footer.php"); ?>
```

Probeer maar uit! surf naar users.php

Resultaat:



In onze database hebben we momenteel geen image voor de users. Daarnaast dienen we de User.php class ook aan te passen en een property hiervoor toe te voegen.
Open User.php en pas als volgt aan:

```php
class User extends Dbobject
{
    /**VARIABELEN**/
    protected static $db_table = "users";
    protected static $db_table_fields = array('username', 'password', 'first_name', 'last_name','user_image');

    public $id;
    public $username;
    public $password;
    public $first_name;
    public $last_name;
    public $user_image;
    public $upload_directory = 'img'.DS.'users';
    public $image_placeholder = 'http://place-hold.it/400x400&text=image';
```

Zorg ervoor dat je in de 'img' map een tweede map aanmaakt met de naam 'users'. Hierin zullen alle foto's komen te staan van onze users.

In **User.php** creëeren we een methode die de locatie van de afbeeldingen bevat:
Wanneer we geen image vinden dan geven we de standaard locatie als default terug, nl. image_placeholder die de link bevat naar de site http://place-hold.it.
In het andere geval wordt de echte image teruggegeven. Momenteel kunnen we dit nog niet ophalen.

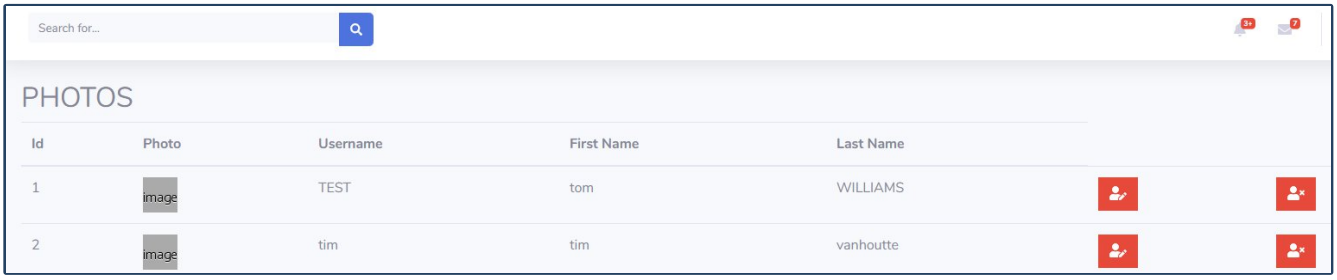
```php
public function image_path_and_placeholder() {
    return empty($this->user_image) ? $this->image_placeholder : $this->upload_directory.DS.$this->user_image;
}
```

Vergeet vervolgens niet om bovenstaande methode in users.php toe te passen.
Open **users.php** en pas volgende aan:
Naast de **methode image_path_and_placeholder** voegen we ook de class **user_image** toe. Deze styled de images naar 40x40 pixels zoals voorzien in styles.css.

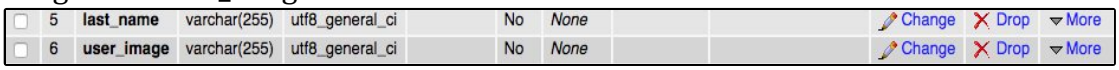
```php
foreach($users as $user): ?>
    <tr>
        <td><?php echo $user->id; ?></td>
        <td><img height="40" width="40" src="<?php echo $user->image_path_and_placeholder(); ?>" alt=""></td>
```

Surf nu naar de pagina van users.php. De default images worden weergegeven

Resultaat:



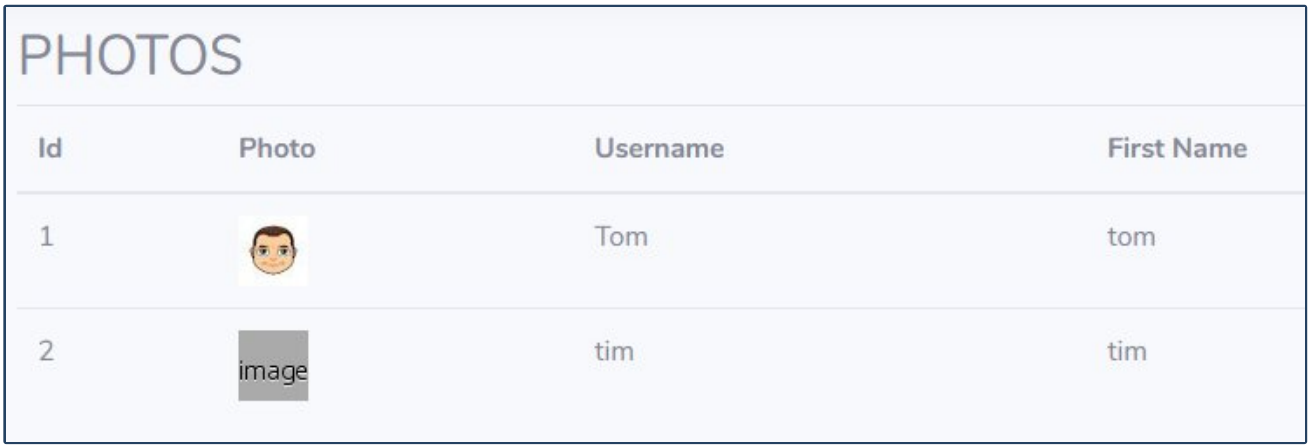
Voeg nu de user_image toe in de tabel van users in de database



Zorg voor een user image die je in de map afbeeldingen plaatst. Neem vervolgens de bestandsnaam (in dit voorbeeld funavatar.jpg) en plaats deze hardcoded bij één van de users in de database.



Surf nu maar naar de pagina alles zal werken.



5.7.1. PAGINA: USER VERWIJDEREN

We hadden reeds een pagina gemaakt die hier op lijkt, nl. delete_Photo.php.
Kopieer de inhoud daarvan in een nieuwe pagina delete_User.php en pas als volgt aan:

```php
<?php include("includes/init.php"); ?>
<?php if(!$session->is_signed_in()) {redirect( location: "login.php");} ?>
<?php

if(empty($_GET['id'])) {
    redirect( location: "users.php");
}

$user = User::find_by_id($_GET['id']);
if($user) {
    $user->delete();
    redirect( location: "users.php");
} else {
    redirect( location: "users.php");
}
?>
```

Probeer maar uit, je kan nu een user verwijderen!

5.7.2. PAGINA: USER TOEVOEGEN

In de **users.php** pagina voegen we een button toe die ons naar de pagina zal brengen om een nieuwe user toe te voegen.

```php
<div class="col-12">
    <h2>PHOTOS</h2>
    <td><a class="btn btn-primary rounded-0" href="add_user.php"><i class="fas fa-user-plus"></i>Add User</a></td>
    <table class="table table-header">
        <thead>
```

We dienen een nieuwe pagina aan te maken, nl. admin/**add_User.php**.
Om dit vlugger uit te voeren **kopieer** je alles van **edit_Photo.php** in de nieuwe pagina **add_User.php.**

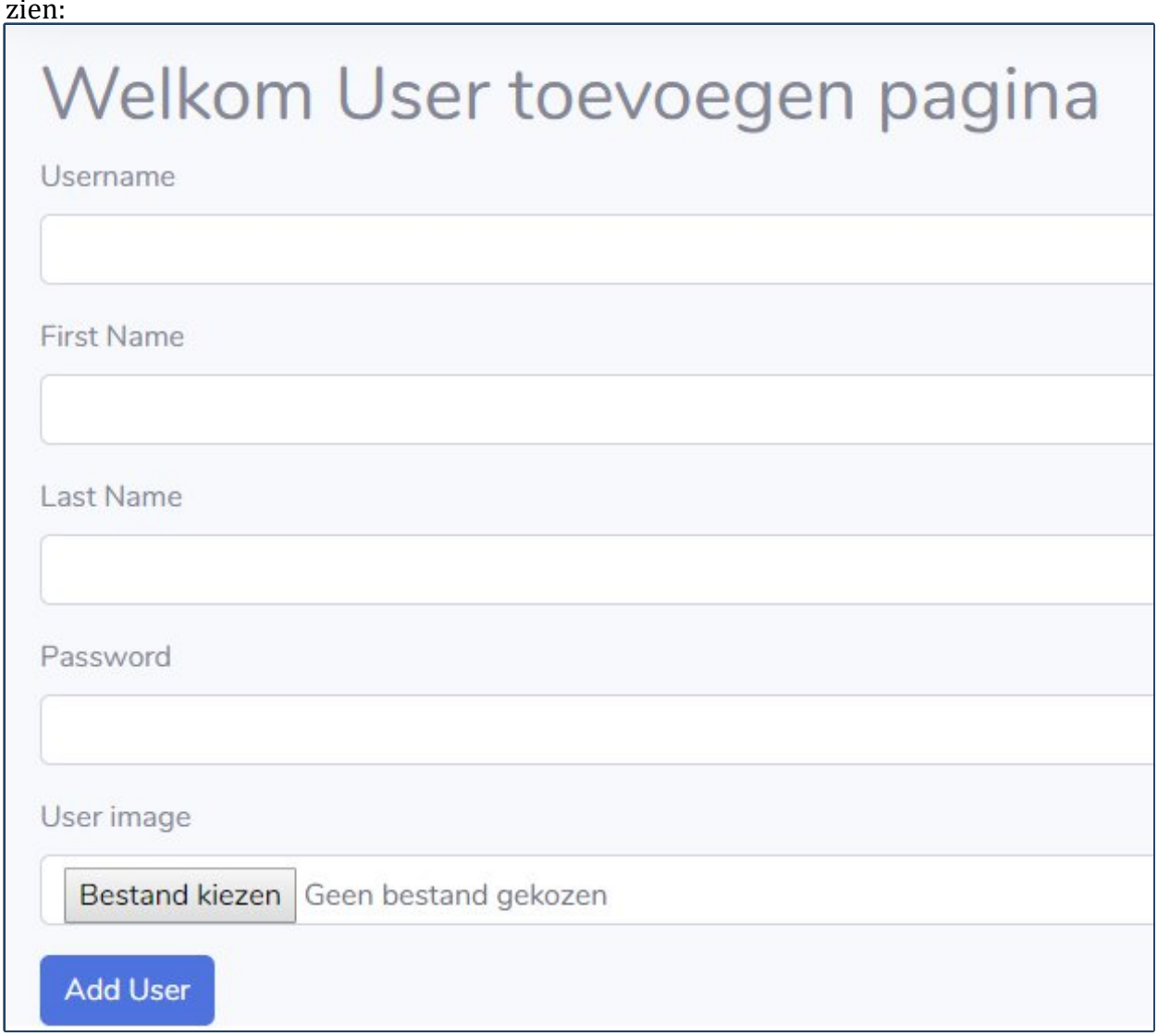
Doe een Find/Replace **photo** naar **user**.
Pas onderstaande code als volgt aan en zet dit even in commentaar:

```php
<?php
include("includes/header.php");
if (!$session->is_signed_in()) {
    redirect( location: "login.php");


}
/*if (empty($_GET['id'])) {
    redirect('users.php');
}else{
    $user = user::find_by_id($_GET['id']);
    if(isset($_POST['update'])){
        if($user){
            $user->title = $_POST['title'];
            $user->caption = $_POST['caption'];
            $user->description = $_POST['description'];
            $user->alternate_text = $_POST['alternate_text'];
            $user->type = $_POST['type'];
            $user->size = $_POST['size'];
            $user->update();
        }
    }
}*/
```

Wijzig de form tag als volgt:

```
?>
<?php include("includes/sidebar.php"); ?>
<?php include("includes/content-top.php"); ?>
<div class="container-fluid">
    <div class="row">
        <div class="col-12">
            <h1>Welkom User toevoegen pagina</h1>
            <form action="add_user.php" method="post" enctype="multipart/form-data">
                <div class="row">
                    <div class="col-12">
                        <div class="form-group">
                            <label for="username">Username</label>
                            <input type="text" name="username" class="form-control">
                        </div>
                        <div class="form-group">
                            <label for="first name">First Name</label>
                            <input type="text" name="first_name" class="form-control">
                        </div>
                        <div class="form-group">
                            <label for="last name">Last Name</label>
                            <input type="text" name="last_name" class="form-control">
                        </div>
                        <div class="form-group">
                            <label for="password">Password</label>
                            <input type="password" name="password" class="form-control">
                        </div>
                        <div class="form-group">
                            <label for="file">User image</label>
                            <input type="file" name="user_image" class="form-control">
                        </div>
                        <input type="submit" name="add_user" value="Add User" class="btn btn-primary">
                    </div>
                </div>
            </form>
        </div>
    </div>
</div>
<?php include("includes/footer.php"); ?>
```

Wanneer je nu naar users.php surft en op de "add user" button klikt dan dien je reeds het volgende te zien:



In add_User.php wijzigen we onderstaande code:
we creëeren een nieuwe instantie van het object user met de naam $user.
Wanneer er op de button create user wordt gedrukt zullen alle values van het formulier gepost worden in ons object $user.

```php
<?php
include("includes/header.php");
if (!$session->is_signed_in()) {
    redirect( location: "login.php");

}
$user = new User();
if(isset($_POST['submit'])){
    if($user){
        $user->username = $_POST['username'];
        $user->first_name = $_POST['first_name'];
        $user->last_name = $_POST['last_name'];
        $user->password = $_POST['password'];
        $user->save();
    }
}
```

Om bovenstaande uit te voeren hadden we reeds code geschreven die we kunnen gebruiken uit **Photo.php**.

Kopieer de **set_file method** en plak deze in **User.php.**

```php
public function set_file( $file ) {
    echo $file;

    if ( empty( $file ) || ! $file || ! is_array( $file ) ) {
        $this->errors[] = "There was no file uploaded here";

        return false;

    } elseif ( $file['error'] != 0 ) {

        $this->errors[] = $this->upload_errors_array[ $file['error'] ];

        return false;

    } else {

        $this->user_image = basename( $file['name'] );
        $this->tmp_path = $file['tmp_name'];
        $this->type     = $file['type'];
        $this->size     = $file['size'];

    }
}
```

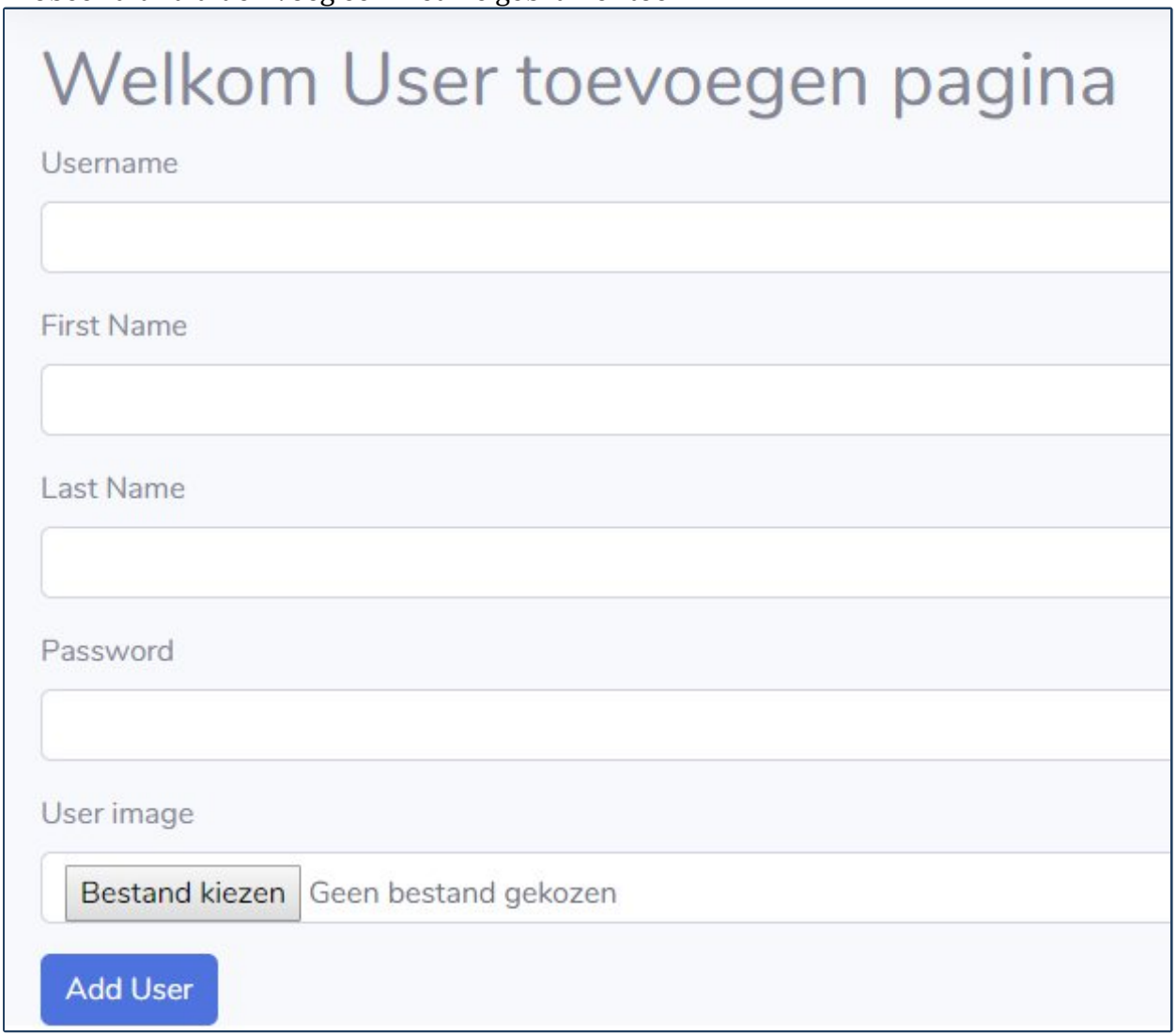
Kopieer daarna de **save methode** uit **Photo.php** in plak deze in **User.php.**
Hernoem de **save** methode naar **save_user_and_image().** Deze methode zal controleren of de gebruiker bestaat en tegelijk deze updaten. Indien niet zal hij deze creëeren.
Doe een zoek en vervang van **filename** door **user_image** om de naamgeving gelijk te houden.
Daarnaast zullen we ook de locatie van de map img/users meegeven aan de variabele $target_path.
De functie move_uploaded_file zal zorgen dat deze beward wordt op de locatie van $target_path.

```
public function save_user_and_image(){
    $target_path = SITE_ROOT . DS . "admin" . DS . $this->upload_directory . DS . $this->user_image;
    if($this->id){
        move_uploaded_file($this->tmp_path, $target_path);
        $this->update();
        unset($this->tmp_path);
        return true;
    }else{
        if(!empty($this->errors)){
            return false;
        }
        if(empty($this->user_image) || empty($this->tmp_path)){
            $this->errors[]= "File not available";
            return false;
        }
        if (file_exists($target_path)){
            $this->errors[] = "File {$this->user_image} exists!";
            return false;
        }
        if(move_uploaded_file($this->tmp_path, $target_path)){
            if($this->create()){
                unset($this->tmp_path);
                return true;
            }
        }
        else{
            $this->errors[]= "This folder has no write rights!";
            return false;
        }
    }
}
```

Open vervolgens **add_User.php** en pas de velden en methode aan:

```
<?php
include("includes/header.php");
if (!$session->is_signed_in()) {
    redirect( location: "login.php");

}
$user = new User();
if(isset($_POST['submit'])){
    if($user){
        $user->username = $_POST['username'];
        $user->first_name = $_POST['first_name'];
        $user->last_name = $_POST['last_name'];
        $user->password = $_POST['password'];
        $user->set_file($_FILES['user_image']);
        $user->save_user_and_image();
    }
}
```

Probeer dit nu uit en voeg een nieuwe gebruiker toe!



Controleer je tabel users in je database. De gebruiker werd toegevoegd.

5.7.3. PAGINA: USER WIJZIGEN

Wanneer we surfen naar de users pagina, krijgen we een overzicht van alle users.

Id	Photo	Username	First Name	Last Name
1		Tom	Tom	WILLIAMS
	Delete Edit View			

Bij het klikken op de edit link wordt het id van deze user meegegeven. Dit id dienen we op te vangen in **edit.User.php** die we nog dienen aan te maken.

Gebruik de **add_User.php** pagina en bewaar deze als **edit_User.php.** We dienen ervoor te zorgen dat deze pagina enkel kan aangesproken worden door beheerders (administrators) .

```php
<?php
include("includes/header.php");
if (!$session->is_signed_in()) {
    redirect( location: "login.php");

}
if(empty($_GET['id'])){
    redirect( location: "users.php");
}
$user = User::find_by_id($_GET['id']);
if(isset($_POST['update_user'])){
    if($user){
        $user->username = $_POST['username'];
        $user->first_name = $_POST['first_name'];
        $user->last_name = $_POST['last_name'];
        $user->password = $_POST['password'];
        $user->set_file($_FILES['user_image']);
        $user->save_user_and_image();
    }
}
?>
```

De methode User::find_by_id haalt het id uit de url.

We passen nu het formulier aan zodat de waarden uit onze tabel users worden ingelezen in het formulier.

```
<div class="container-fluid">
    <div class="row">
        <div class="col-12">
            <h1>Welkom User wijzigen pagina</h1>

            <form action="" method="post" enctype="multipart/form-data">
                <div class="row">
                    <div class="col-12">
                            <div class="form-group">
                            <label for="username">Username</label>
                                <input type="text" name="username" class="form-control"
                                        value="<?php echo $user->username; ?>"
                        </div>
                        <div class="form-group">
                            <label for="first name">First Name</label>
                            <input type="text" name="first_name" class="form-control"
                                    value="<?php echo $user->first_name; ?>"
                        </div>
                        <div class="form-group">
                            <label for="last name">Last Name</label>
                            <input type="text" name="last_name" class="form-control"
                                    value="<?php echo $user->last_name; ?>"
                        </div>
                        <div class="form-group">
                            <label for="password">Password</label>
                            <input type="password" name="password" class="form-control"
                                    value="<?php echo $user->password; ?>"
                        </div>
                        <div class="form-group">
                            <img src="<?php echo $user->image_path_and_placeholder(); ?>"
                                alt="" class="img-fluid" width="40" height="40">
                            <label for="file">User image</label>
                            <input type="file" name="user_image" class="form-control">
                        </div>
                        <input type="submit" name="update_user" value="Update User" class="btn btn-primary">
                    </div>
                </div>
            </form>
        </div>
    </div>
</div>
<?php include("includes/footer.php"); ?>
```

Er is nog één probleem die we dienen op te lossen namelijk het updaten van een foto.
Wanneer we in User.php pagina op edit klikken dan krijgen we perfect alle gegevens met foto te zien.
Wanneer we echter de foto wijzigen en op update klikken dan zouden we de foto niet zien! We wijzigen enkel de naam maar niet de foto zelf in de pagina.

Wanneer we in **User.php** naar onderstaande code kijken dan proberen we eigenlijk een lege image te updaten, wat natuurlijk niet werkt. Dit hadden we reeds opgevangen (zie screenshot hieronder).

```
if($this->id){
    move_uploaded_file($this->tmp_path, $target_path);
    $this->update();
    unset($this->tmp_path);
    return true;
```

Dit lossen we nu als volgt op:
Wis bovenstaande code (blauw gekleurd) en vergeet onderaan ook niet de afsluitende **curly bracket te schrappen**.

Open nu **edit_User.php** en pas als volgt aan:

```
$user = User::find_by_id($_GET['id']);
if(isset($_POST['update_user'])){
    if($user){
        $user->username = $_POST['username'];
        $user->first_name = $_POST['first_name'];
        $user->last_name = $_POST['last_name'];
        $user->password = $_POST['password'];
        if (empty($_FILES['user_image'])) {
            $user->save();
        } else {
            $user->set_file($_FILES['user_image']);
            $user->save_user_and_image();
            $user->save();
            redirect( location: "edit_user.php?id={$user->id}");
        }
    }
}
?>
```

Probeer nu maar uit! Laad een nieuwe user op en probeer data aan te passen.

Wanneer je een error zou krijgen willen we weten over welke error dit gaat. In **Photo.php** hadden we dit opgevangen. Laten we dit beschikbaar maken voor de volledige site. **Knip** en **plak** onderstaande code naar **Db_object.php**.

```
public $errors = array();
public $upload_errors_array = array(

    UPLOAD_ERR_OK         => "There is no error",
    UPLOAD_ERR_INI_SIZE   => "The uploaded file exceeds the upload_max_filesize directive in php.ini",
    UPLOAD_ERR_FORM_SIZE  => "The uploaded file exceeds the MAX_FILE_SIZE directive that was specified in the HTML form",
    UPLOAD_ERR_PARTIAL    => "The uploaded file was only partially uploaded.",
    UPLOAD_ERR_NO_FILE    => "No file was uploaded.",
    UPLOAD_ERR_NO_TMP_DIR => "Missing a temporary folder.",
    UPLOAD_ERR_CANT_WRITE => "Failed to write file to disk.",
    UPLOAD_ERR_EXTENSION  => "A PHP extension stopped the file upload."
);
```

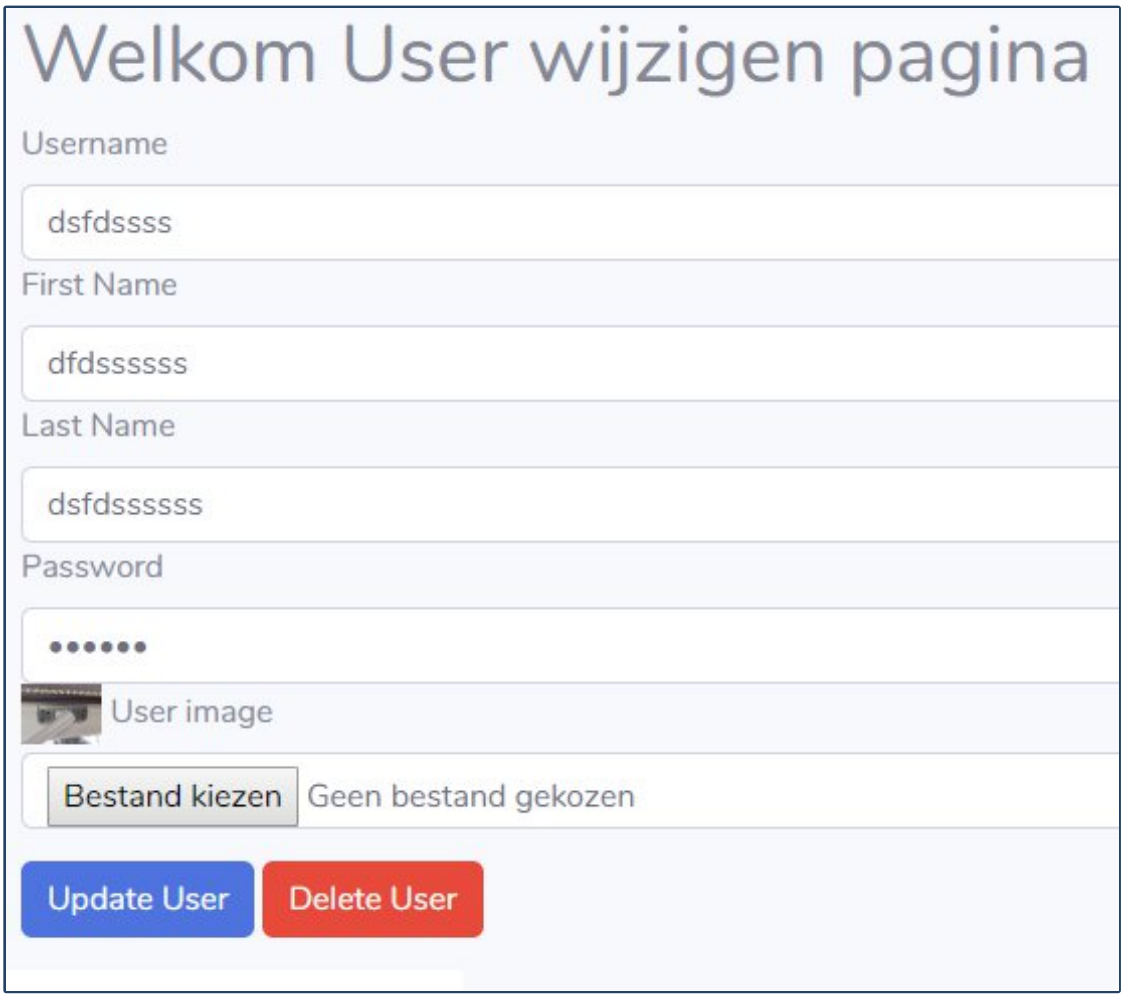
5.7.4. PAGINA: USER VERWIJDEREN

We voegen hiervoor een delete button toe aan **edit_User.php:**

```
<input type="submit" name="update_user" value="Update User" class="btn btn-primary">
<a class="btn btn-danger" href="delete_user.php?id=<?php echo $user->id; ?>">Delete User</a>
```

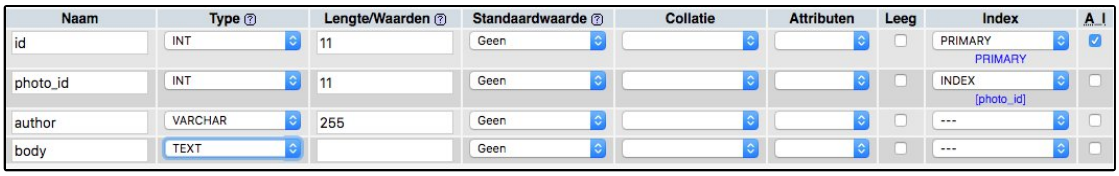
We hadden de pagina delete_User.php reeds aangemaakt. Deze button zal dus reeds werken en de gebruiker verwijderen.

Resultaat:



5.8. COMMENTS

Voor het onderdeel commentaar hebben we nog geen tabel voorzien in onze database. Deze gaan we nu eerst toevoegen.
Het is de bedoeling dat we per foto commentaar kunnen toevoegen.
Maak een nieuwe tabel **comments** aan en voeg de volgende velden toe:
we voegen ook een veld photo_id toe die zorgt voor de relatie met onze photos tabel. Om het zoeken te versnellen door phpmyadmin kunnen we deze FK (foreign key) op index zetten. Index versnelt het zoeken in databases.



Naam	Type	Lengte/Waarden	Standaardwaarde	Collatie	Attributen	Leeg	Index	A_I
id	INT	11	Geen				PRIMARY PRIMARY	
photo_id	INT	11	Geen				INDEX [photo_id]	
author	VARCHAR	255	Geen				---	
body	TEXT		Geen				---	

5.9. COMMENT CLASS

We coderen OOP, dus ook hier zullen we ervoor zorgen dat we onze data object georiënteerd aanspreken.
We hebben reeds enkele classes geschreven waaronder User.php.
Open **User.php** en bewaar deze als **Comment.php.**

Pas nu comments.php aan als volgt en wis alle andere methodes:

```php
<?php
class Comment extends Db_object {
    protected static $db_table = "comments";
    protected static $db_table_fields = array('id', 'photo_id','author', 'body');
    public $id;
    public $photo_id;
    public $author;
    public $body;

}
?>
```

We zullen nu enkele methodes schrijven die we nodig zullen hebben in onze applicatie met betrekkening tot het creëeren/opzoeken van comments.
De volgende stap is nu een eerste methode te creëeren die ervoor zorgt dat we comments kunnen opslaan. In onderstaande methode **create_comment** zullen er 3 parameters binnenkomen. Het if statement zorgt ervoor dat we ons comment object ENKEL VULLEN wanneer deze 3 parameters NIET LEEG zijn.

```php
<?php
class Comment extends Db_object {
    protected static $db_table = "comments";
    protected static $db_table_fields = array('id', 'photo_id','author', 'body');
    public $id;
    public $photo_id;
    public $author;
    public $body;

    public static function create_comment($photo_id, $author="Test", $body=""){
        if (!empty($photo_id) && !empty($author) && !empty($body)){
            $comment = new Comment();
            $comment->photo_id = (int)$photo_id;
            $comment->author = $author;
            $comment->body = $body;

            return $comment;
        }else{
            return false;
        }
    }// end create_comment

}// end class

?>
```

Een tweede methode die we zullen schrijven is een methode die de comments gerelateerd aan een photo zal opzoeken.
Hiervoor heeft deze functie natuurlijk een photo_id nodig als parameter.
Aan de hand van deze photo_id kunnen we uit de tabel comments als comments opzoeken gelinkt aan deze photo. Merk op dat we .self::$db_table gebruiken om de tabelnaam comments weer te geven. Herinner je dat dit enkel kan wanneer je een functie als static beschouwd. Self zorgt er dan voor dat je de naam van de class zelf als variabele kan gebruiken!
De query ($sql) wordt dan uiteindelijk uitgevoerd door de methode find_by_query uit de class Db_object.php die we kunnen bereiken dankzij de extends property toegevoegd aan de class van comments.

```php
}//end create comment

public static function find_the_comments($photo_id){
    global $database;
    $sql = "SELECT * FROM " .self::$db_table;
    $sql .= " WHERE photo_id = " . $database->escape_string($photo_id);
    $sql .= " ORDER BY photo_id ASC";

    return self::find_this_query($sql);
}
```

Vergeet ook niet om Comment.php toe te voegen aan het init.php bestand.

```php
<?php
    defined( name: 'DS') ? null : define('DS', DIRECTORY_SEPARATOR);
    define('SITE_ROOT', 'D:' . DS . 'Tom' . DS . 'www' . DS . 'blog');
    defined( name: 'INCLUDES_PATH') ? null : define('INCLUDES_PATH', SITE_ROOT.DS.'admin'.DS.'includes');
    defined( name: 'IMAGES_PATH') ? null : define('IMAGES_PATH', SITE_ROOT.DS.'admin'.DS.'img');

    require_once(INCLUDES_PATH.DS."functions.php");
    require_once(INCLUDES_PATH.DS."config.php");
    require_once(INCLUDES_PATH.DS."Database.php");
    require_once(INCLUDES_PATH.DS."Db_object.php");
    require_once(INCLUDES_PATH.DS."User.php");
    require_once(INCLUDES_PATH.DS."Photo.php");
    require_once(INCLUDES_PATH.DS."Session.php");
    require_once(INCLUDES_PATH.DS."Comment.php");
?>
```

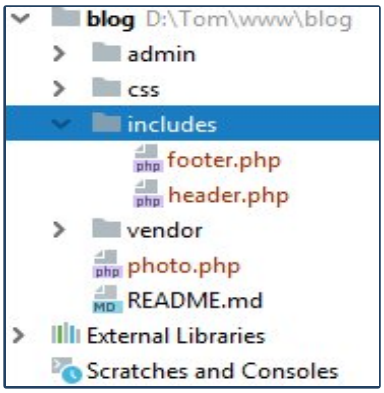
5.10. COMMENTS FORMULIER

5.11. COMMENTAAR MAKEN/TONEN IN DE FRONT-END

Wanneer we nu de methodes en de comments class willen toepassen dan kunnen we dit testen op de volgende locatie in onze site:
http://localhost/blog/Photo.php. Deze pagina bestaat nog niet. Ook hier zullen we gebruik maken van een een template. Download de template van de volgende link die we als front-end zullen gebruiken.
https://startbootstrap.com/previews/blog-post/

Wanneer je deze pagina bekijkt dan zie je enkele statische commentaarlijnen gelinkt aan een blog bericht met foto.
Dit dienen we dynamisch te maken.

Unzip nu het gedownloade bestand en voeg de volgende aanpassingen uit:



Zoals je hierboven kan zien dien je de volledige css map en de vendor map van het uitgepakte bestand in de root van je blog te kopiëren.
Ook in de root van je blog maak je je startpagina aan: **Photo.php**. Hier zal één blog item met zijn commentaren worden weergegeven.
Daarnaast maak je in de root een **includes map** aan met daarin **footer.php** en **header.php.**

In het uitgepakte bestand zal je index.html terugvinden. Deze zullen we nu splitsen over photo .php, footer.php en header.php.

Header.php:

```
<html lang="en">

<head>

    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="description" content="">
    <meta name="author" content="">

    <title>Blog Post - Start Bootstrap Template</title>

    <!-- Bootstrap core CSS -->
    <link href="vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="css/blog-post.css" rel="stylesheet">

</head>

<body>

<!-- Navigation -->
<nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">
    <div class="container">
        <a class="navbar-brand" href="#">Start Bootstrap</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive"
                aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarResponsive">
            <ul class="navbar-nav ml-auto">
                <li class="nav-item active">
                    <a class="nav-link" href="#">Home
                        <span class="sr-only">(current)</span>
                    </a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#">About</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#">Services</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#">Contact</a>
                </li>
            </ul>
        </div>
    </div>
</nav>
```

In footer.php:

```
<!-- Footer -->
<footer class="py-5 bg-dark">
    <div class="container">
        <p class="m-0 text-center text-white">Copyright © Your Website 2019</p>
    </div>
    <!-- /.container -->
</footer>

<!-- Bootstrap core JavaScript -->
<script src="vendor/jquery/jquery.min.js"></script>
<script src="vendor/bootstrap/js/bootstrap.bundle.min.js"></script>

</body>

</html>
```

Open nu in **de front-end** van de site **Photo.php** en voeg de volledige rubriek vanaf Page Content tot juist voor de footer hier toe.

```
<!-- Page Content -->
<div class="container">

    <div class="row">

        <!-- Post Content Column -->
        <div class="col-lg-8">

            <!-- Title -->
            <h1 class="mt-4">Post Title</h1>

            <!-- Author -->
            <p class="lead">
                by
                <a href="#">Start Bootstrap</a>
            </p>

            <hr>

            <!-- Date/Time -->
            <p>Posted on January 1, 2019 at 12:00 PM</p>

            <hr>

            <!-- Preview Image -->
            <img class="img-fluid rounded" src="http://placehold.it/900x300" alt="">

            <hr>

            <!-- Post Content -->
            <p class="lead">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Du

            <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur nat

            <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Eos, doloribus,

            <blockquote class="blockquote">
                <p class="mb-0">Lorem ipsum dolor sit amet, consectetur adipiscing elit.
                <footer class="blockquote-footer">Someone famous in
                    <cite title="Source Title">Source Title</cite>
                </footer>
            </blockquote>

            <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Error, nostrum,

            <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Harum, dolor qu

            <hr>
```

Include nu ook de header en de footer in photo .php. Daarnaast moeten we ook de data kunnen inlezen vanaf het backend gedeelte. Voeg hiervoor ook init.php toe.

```
<?php
include("includes/header.php");
require_once("admin/includes/init.php");
?>
```

```
    </div>
    <!-- /.row -->

</div>
<?php include("includes/footer.php"); ?>
```

Pas vervolgens het init.php bestand in de backend ook aan als volgt.

```php
<?php
    defined( name: 'DS') ? null : define('DS', DIRECTORY_SEPARATOR);
    define('SITE_ROOT', 'D:' . DS . 'Tom' . DS . 'www' . DS . 'blog');
    defined( name: 'INCLUDES_PATH') ? null : define('INCLUDES_PATH', SITE_ROOT.DS.'admin'.DS.'includes');
    defined( name: 'IMAGES_PATH') ? null : define('IMAGES_PATH', SITE_ROOT.DS.'admin'.DS.'img');

    require_once(INCLUDES_PATH.DS."functions.php");
    require_once (INCLUDES_PATH.DS."config.php");
    require_once(INCLUDES_PATH.DS."Database.php");
    require_once(INCLUDES_PATH.DS."Db_object.php");
    require_once(INCLUDES_PATH.DS."User.php");
    require_once(INCLUDES_PATH.DS."Photo.php");
    require_once(INCLUDES_PATH.DS."Session.php");
?>
```

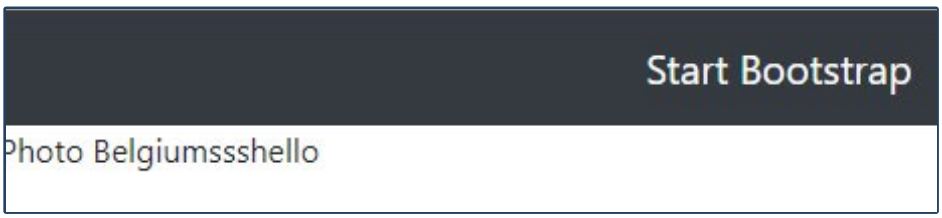
Pas nu het formulier voor het commentaar als volgt aan in Photo.php:

```php
<!-- Comments Form -->
<div class="card my-4">
    <h5 class="card-header">Leave a Comment:</h5>
    <div class="card-body">
        <form method="post">
            <div class="form-group">
                <label for="author">Author</label>
                <input type="text" name="author" class="form-control">
            </div>
            <div class="form-group">
                <textarea class="form-control" rows="3" name="body"></textarea>
            </div>
            <button type="submit" name="submit" class="btn btn-primary">Submit</button>
        </form>
    </div>
</div>
```

Open nu Photo.php in de front-end en pas als volgt aan:
probeer eerste de submit knop uit door te surfen naar localhost/blog/Photo.php en druk op de submit knop. De tekst hello komt op het scherm.
Probeer vervolgens de titel van een object op te vragen uit de database door een object te vullen en de titel op het scherm weer te geven.
Doe dit door te surfen naar: **http://localhost/blog/Photo.php?id=1** (zorg wel dat dit id in de database bestaat)

```php
<?php
    require_once("admin/includes/init.php");
    $photo = Photo::find_by_id($_GET['id']);
    echo $photo->title;

    if(isset($_POST['submit'])){
        echo "hello";
    }
?>
```

Titel en Hello worden weergegeven (zie screenshot hieronder)



Wanneer er geen id aanwezig is in de url, dan vangen we dit op door onmiddellijk te redirecten naar de index.php pagina.

```php
<?php
    require_once("admin/includes/init.php");

    if(empty($_GET['id'])){
        redirect( location: "index.php");
    }
    $photo = Photo::find_by_id($_GET['id']);

    if(isset($_POST['submit'])){
        echo "hello";
    }
?>
```

Aangezien we in de **root van onze blog nog geen index.php** hebben maken we die eerst aan. Dit zal een simpel bestand zijn met de verwijziging naar onze **index.php** (home pagina).

```php
<?php
include("includes/header.php");
?>
<div class="container-fluid">
    <div class="row">
        <div class="col-12">
            <h1>Dit is de homepagina van het frontend gedeelte!</h1>
        </div>
    </div>
</div>
<?php
include("includes/footer.php");
?>
```

Om te zorgen dat de footer mooi beneden blijft staan in de layout voeg de bootstrap 4 class f**ixed-bottom** toe.
Open includes/footer.php:

```php
<!-- Footer -->
<footer class="py-5 bg-dark fixed-bottom">
    <div class="container">
        <p class="m-0 text-center text-white">Copyright © Your Website 2019</p>
    </div>
    <!-- /.container -->
</footer>

<!-- Bootstrap core JavaScript -->
<script src="vendor/jquery/jquery.min.js"></script>
<script src="vendor/bootstrap/js/bootstrap.bundle.min.js"></script>

</body>

</html>
```

Resultaat:



Nu dienen we nog de data uit het comment form te halen en deze weg te schrijven naar onze database. We halen dus het id van de photo uit de url op want dit hebben we nodig om het commentaar aan de photo te koppelen.
Zoals je hieronder kan zien controleren we met het if statement of er wel degelijk een comment aanwezig is en deze gelijktijdig te bewaren en te tonen door te redirecten naar photo pagina.
Open Photo.php en pas aan als volgt:

```php
<?php
    require_once("admin/includes/init.php");

    if(empty($_GET['id'])){
        redirect( location: "index.php");
    }
    $photo = Photo::find_by_id($_GET['id']);

    if(isset($_POST['submit'])){
        $author = trim($_POST['author']);
        $body = trim($_POST['body']);

        $new_comment = Comment::create_comment($photo->id, $author, $body);

        if($new_comment && $new_comment->save()) {
            redirect( location: "photo.php?id={$photo->id}");
        } else {
            $message = "There are some problems saving";
        }
    } else {
        $author = "";
        $body = "";
    }

    $comments = Comment::find_the_comments($photo->id);
?>
```

We zullen enkel met single comments werken. D.w.z. dat 1 photo meerdere commentaren kan bevatten. We **houden** dus de **single comment** over en de **nested comments** in onze template **verwijderen** we. Wijzig nu de comments als volgt:

```
        <?php foreach($comments as $comment): ?>
        <!-- Single Comment -->
        <div class="media mb-4">
            <img class="d-flex mr-3 rounded-circle" src="http://placehold.it/50x50" alt="">
            <div class="media-body">
                <h5 class="mt-0"><?php echo $comment->author; ?></h5>
                    <p><?php echo $comment->body; ?></p>
            </div>
        </div>

        <?php endforeach; ?>

    </div>
```

Vooraleer we dit kunnen testen dienen we minstens 1 commentaarlijn in onze database toe te voegen in de tabel comments.



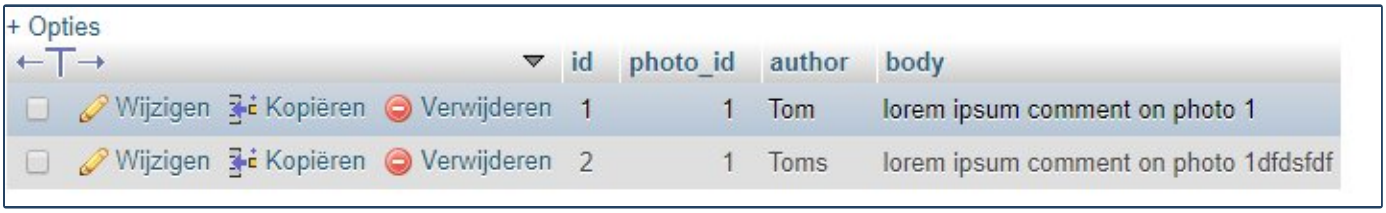
Surf naar http://localhost/blog/Photo.php?id=1?
Dit zou moeten werken.

Resultaat:



Wanneer je een tweede wijziging zou uitvoeren dat zal je voor deze foto 2 commentaarlijnen staan hebben. **Gebruik nu het submit formulier en bekijk je database**.

Resultaat:
Zoals je hieronder kan zien zijn er voor dit photo_id 2 commentaarlijnen.



Resultaat:


Leave a Comment:
Author
Submit
Tom
lorem ipsum comment on photo 1
Toms
lorem ipsum comment on photo 1dfdsfdf

Nu kunnen we in principe deze pagina ook voorzien van alle invoer van de photo tabel zelf ook.

```
<!-- Post Content Column -->
<div class="col-lg-8">

    <!-- Title -->
    <h1 class="mt-4"><?php echo $photo->title; ?></h1>
    <!-- Author -->
    <p class="lead">
        by
        <a href="#">Start Bootstrap</a>
    </p>
    <hr>
    <!-- Date/Time -->
    <p>Posted on January 1, 2019 at 12:00 PM</p>
    <hr>
    <!-- Preview Image -->
    <img class="img-fluid rounded" src="<?php echo 'admin'.DS.$photo->picture_path(); ?>" width="900" height="300" alt="">
    <hr>
    <!-- Post Content -->
    <p>
        <?php echo $photo->description; ?>
    </p>
    <hr>
    <!-- Comments Form -->
```

Resultaat:



Nu heb je reeds een idee hoe je een één-op-veel relatie hebt gemaakt, nl. één photo kan meerdere commentaren bevatten. Je kan dit uitbreiden zoals je natuurlijk zelf wil. Eventueel kan je de database uitbreiden met een relatie tussen de user die de foto heeft genomen.

5.12. COMMENTAAR MAKEN/TONEN IN DE BACK-END

De bedoeling hier is het tonen van **alle** commentaren van **alle** photos.
Er werd natuurlijk commentaar toegevoegd door gebruikers. Het zou interessant zijn moesten we dit in de backend ook kunnen opvangen om bijvoorbeeld ervoor te zorgen dat enkele interessante commentaren worden getoond.

Kopieer alles van **users.php** in **comments.php** en pas als volgt aan:
vervang de class user naar comment:

```
$comments = Comment::find_all();
```

Doe een Find/Replace van **users** naar **comments.**
Doe een Find/Replace van **user** naar **comment**.

```
<?php
    include("includes/header.php");
    if(!$session->is_signed_in()){
        redirect( location: 'login.php');
    }
    $comments = Comment::find_all();
?>
<?php include("includes/sidebar.php"); ?>
<?php include("includes/content-top.php"); ?>
<div class="container-fluid">
    <div class="row">
        <div class="col-12">
            <h2>PHOTOS</h2>
            <td><a class="btn btn-primary rounded-0" href="add_comment.php"><i class="fas fa-comment-plus"></i>Add comment</a></td>
            <table class="table table-header">
                <thead>
                <tr>
                    <th>Id</th>
                    <th>Author</th>
                    <th>Body</th>
                </tr>
                </thead>
                <tbody>
                <?php
                foreach($comments as $comment): ?>
                    <tr>
                        <td><?php echo $comment->id; ?></td>
                        <td><?php echo $comment->author; ?></td>
                        <td><?php echo $comment->body; ?></td>
                        <td><a class="btn btn-danger rounded-0" href="delete_comment.php?id=<?php echo
                            $comment->id; ?>"><i class="far fa-trash-alt"></i></a></td>

                    </tr>
                <?php endforeach; ?>
                </tbody>
            </table>
        </div>

    </div>
</div>
<?php include("includes/footer.php"); ?>
```

In localhost/blog/admin/comments.php zou je nu je commentaarlijnen moeten zien staan.

5.12.1. COMMENTAAR DELETEN IN DE BACK-END

We hebben dit reeds geschreven **in delete_User.php**. **Bewaar** dit **als delete_Comment.php**.

Vervang als volgt :

```php
<?php include("includes/init.php"); ?>
<?php
if(!$session->is_signed_in()){
    redirect( location: "login.php");

}
if(empty($_GET['id'])){
    redirect( location: 'comments.php');
}

$comment = Comment::find_by_id($_GET['id']);
if($comment){
    $comment->delete();
    redirect( location: 'comments.php');
}else{
    redirect( location: 'comments.php');
}
?>
```

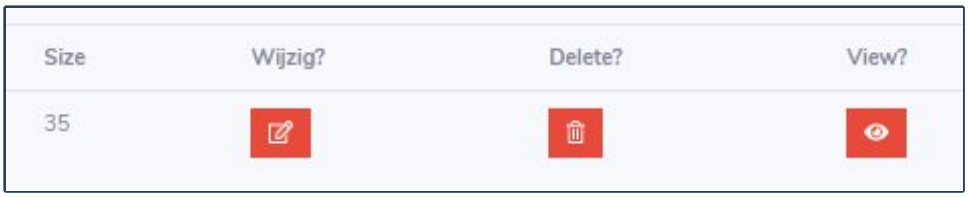
Doe een Find/Replace van **users** naar **comments.**
Doe een Find/Replace van **user** naar **comment**.

Surf nu naar localhost/blog/admin/comments.php en druk op de delete link. De commentaarlijn is nu verwijderd.

Open nu **photos.php** en pas de **view** link aan. Deze link dient ons naar de detail van de photo te brengen waar we een **commentaarlijn** zullen **toevoegen**.

```php
    <th>View?</th>

</tr>
</thead>
<tbody>
<?php
foreach($photos as $photo): ?>
    <tr>

        <td><img  src="<?php echo $photo->picture_path(); ?>" height="40"
                  width="40"
                  alt=""></td>
        <td class="d-flex align-self-stretch"><?php echo $photo->id; ?></td>
        <td><?php echo $photo->title; ?></td>
        <td><?php echo $photo->caption; ?></td>
        <td><?php echo $photo->filename; ?></td>
        <td><?php echo $photo->alternate_text; ?></td>
        <td><?php echo $photo->size; ?></td>
        <td><a class="btn btn-danger rounded-0" href="edit_photo.php?id=<?php echo
            $photo->id; ?>"><i class="far fa-edit"></i></a></td>
        <td><a class="btn btn-danger rounded-0" href="delete_photo.php?id=<?php echo
            $photo->id; ?>"><i class="far fa-trash-alt"></i></a></td>
        <td><a class="btn btn-danger rounded-0" href="../photo.php?id=<?php echo
            $photo->id; ?>"><i class="fas fa-eye"></i></a></td>
```

Probeer maar uit: http://localhost/blog/admin/photos.php



Het volgende dat we wensen te bewerkstelligen is het toevoegen van commentaarlijnen aan een photo in de backend.
De bedoeling hier is het tonen van **enkel de commentaren van één photo.**
Open nu **comments.php** en bewaar als **comments_Photo.php**

```php
<?php include("includes/header.php"); ?>
<?php
if(!$session->is_signed_in()){
    redirect( location: "login.php");
}
?>
<?php

if(empty($_GET['id'])){
    redirect( location: "photos.php");
}
$comments = Comment::find_the_comments($_GET['id']);
?>
    <!-- Navigation -->
```

```php
<div class="col-12">
    <h2>POSTED COMMENTS FOR THIS PHOTO</h2>
    <td><a class="btn btn-primary rounded-0" href="add_comment.php"><i class="fas fa-comment-medical"></i></i>Add comment</a></td>
    <table class="table table-header">
        <thead>
        <tr>
            <th>Id</th>
            <th>Author</th>
            <th>Body</th>
        </tr>
        </thead>
        <tbody>
        <?php
        foreach($comments as $comment): ?>
            <tr>
                <td><?php echo $comment->id; ?></td>
                <td><?php echo $comment->author; ?></td>
                <td><?php echo $comment->body; ?></td>
                <td><a class="btn btn-danger rounded-0" href="delete_comment.php?id=<?php echo
                    $comment->id; ?>"><i class="far fa-trash-alt"></i></a></td>

            </tr>
        <?php endforeach; ?>
```

Wanneer we nu surfen naar naar **localhost/blog/admin/comments_Photo.php?id=1**, dan zie je de commentaarlijnen staan. (zorg dat je wel enkele commentaarlijnen hebt toegevoegd.

Resultaat:

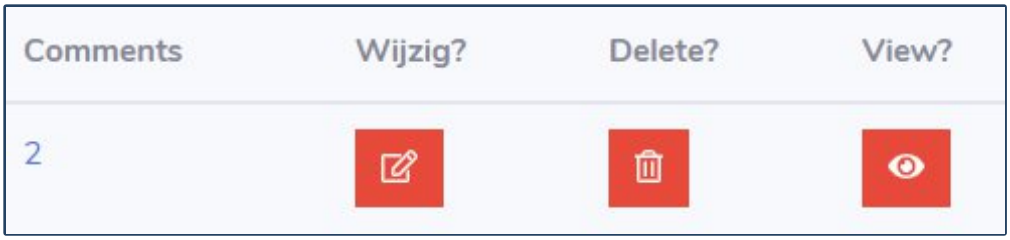


De add comment button doet hier niets momenteel. Je kan natuurlijk zelf altijd voor een formulier zorgen die aan één photo nog een commentaarlijn zou toevoegen.

In **admin/photos.php** zorgen we voor de link en het aantal comments die een photo heeft.
Eerst gaan we het aantal commentaarlijnen toevoegen. Daarnaast zorgen we ervoor wanneer we op het aantal klikken, de commentaarlijnen worden getoond.
Open **photos.php** en pas aan:

```
        <th>Comments</th>
        <th>Wijzig?</th>
        <th>Delete?</th>
        <th>View?</th>

    </tr>
    </thead>
    <tbody>
    <?php
    foreach($photos as $photo): ?>
        <tr>

            <td><img  src="<?php echo $photo->picture_path(); ?>" height="40"
                      width="40"
                      alt=""></td>
            <td class="d-flex align-self-stretch"><?php echo $photo->id; ?></td>
            <td><?php echo $photo->title; ?></td>
            <td><?php echo $photo->caption; ?></td>
            <td><?php echo $photo->filename; ?></td>
            <td><?php echo $photo->alternate_text; ?></td>
            <td><?php echo $photo->size; ?></td>
            <td>
                <a href="comments_photo.php?id=<?php echo $photo->id; ?>">
                <?php
                    $comments = Comment::find_the_comments($photo->id);
                    echo count($comments);
                ?>
                </a>
            </td>
```

Resultaat:



Wanneer we momenteel een commentaarlijn verwijderen dan worden we geredirect naar comments.php. Het is natuurlijk beter om op de pagina te blijven staan, nl. comments_Photo.php?id=1.
Dit passen we aan op de volgende manier:
Open **delete_Comment.php** en bewaar als **delete_comment_Photo.php.**
Pas deze als volgt aan:

```php
$comment = Comment::find_by_id($_GET['id']);
if($comment){
    $comment->delete();
    redirect( location: "comment_photo.php?id={$comment->photo_id}");
}else{
    redirect( location: "comment_photo.php?id={$comment->photo_id}");
}
```

Open nu **comments_Photo.php** en pas aan als volgt:

```php
<?php
foreach($comments as $comment): ?>
    <tr>
        <td><?php echo $comment->id; ?></td>
        <td><?php echo $comment->author; ?></td>
        <td><?php echo $comment->body; ?></td>
        <td><a class="btn btn-danger rounded-0" href="delete_comment_photo.php?id=<?php echo
            $comment->id; ?>"><i class="far fa-trash-alt"></i></a></td>

    </tr>
<?php endforeach; ?>
```

Bij het verwijderen van commentaar blijf je nu op het de photopagina zelf staan.

6. ADMIN DASHBOARD

In dit onderdeel bouwen we het admin dashboard op voor de administrator gebruikers.

6.1. DASHBOARD SNIPPETS

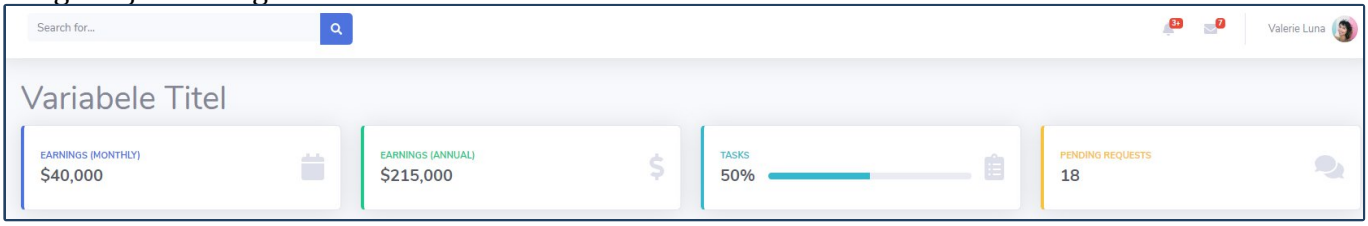
We zullen nu het dashboard vorm geven voor de admin gebruikers. Open includes/content.php en pas als volgt aan:
Uit de html template van ons admin dashboard knippen en plakken we onderstaand gedeelte in content.php




```
<!-- Begin Page Content -->
<div class="container-fluid">
    <div class="row">
        <div class="col-12">
                <h1 class="page-header">
                    Dashboard
                </h1>
                <div class="row">

                <!-- Earnings (Monthly) Card Example -->
                <div class="col-xl-3 col-md-6 mb-4">
                    <div class="card border-left-primary shadow h-100 py-2">
                        <div class="card-body">
                            <div class="row no-gutters align-items-center">
                                <div class="col mr-2">
                                    <div class="text-xs font-weight-bold text-primary text-uppercase mb-1">Earnings (Monthly)</div>
                                    <div class="h5 mb-0 font-weight-bold text-gray-800">$40,000</div>
                                </div>
                                <div class="col-auto">
                                    <i class="fas fa-calendar fa-2x text-gray-300"></i>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>

                <!-- Earnings (Monthly) Card Example -->
                <div class="col-xl-3 col-md-6 mb-4">
                    <div class="card border-left-success shadow h-100 py-2">
                        <div class="card-body">
                            <div class="row no-gutters align-items-center">
                                <div class="col mr-2">
                                    <div class="text-xs font-weight-bold text-success text-uppercase mb-1">Earnings (Annual)</div>
```

Onder de h1 tag pasten we nu de code die je kan downloaden van **dashboard code**.

Dit geeft je het volgende resultaat:



Zorg er nu voor dat dit deel ook dynamisch wordt:

- users met hun aantal
- photos met hun aantal
- comments met hun aantal

Er is één kolom te veel. Die zullen we wissen. Pas ook de titels aan alsmede 3 kolommen volgens de standaarden van bootstrap.

```
<!-- Earnings (Monthly) Card Example -->
<div class="col-12 col-md-4 mb-4">
    <div class="card border-left-primary shadow h-100 py-2">
        <div class="card-body">
            <div class="row no-gutters align-items-center">
                <div class="col mr-2">
                    <div class="text-xs font-weight-bold text-primary text-uppercase mb-1">Users</div>
                    <div class="h5 mb-0 font-weight-bold text-gray-800">$40,000</div>
                </div>
                <div class="col-auto">
                    <i class="fas fa-calendar fa-2x text-gray-300"></i>
                </div>
            </div>
        </div>
    </div>
</div>

<!-- Earnings (Monthly) Card Example -->
<div class="col-12 col-md-4 mb-4">
    <div class="card border-left-success shadow h-100 py-2">
        <div class="card-body">
            <div class="row no-gutters align-items-center">
                <div class="col mr-2">
                    <div class="text-xs font-weight-bold text-success text-uppercase mb-1">Photos</div>
                    <div class="h5 mb-0 font-weight-bold text-gray-800">$215,000</div>
                </div>
                <div class="col-auto">
                    <i class="fas fa-dollar-sign fa-2x text-gray-300"></i>
                </div>
            </div>
        </div>
    </div>
</div>


<!-- Pending Requests Card Example -->
<div class="col-12 col-md-4 mb-4 mb-4">
    <div class="card border-left-warning shadow h-100 py-2">
        <div class="card-body">
            <div class="row no-gutters align-items-center">
```

Resultaat:



Nu dienen we enkel nog de aantal op te halen uit de database.

In admin/index.php pas je aan als volgt:
we halen alle users, comments en photos op en steken die in variabelen.

```php
<?php include("includes/header.php"); ?>
<?php
if(!$session->is_signed_in()){
    redirect( location: "login.php");
}
$aantalUsers = User::find_all();
$aantalComments = Comment::find_all();
$aantalPhotos = Photo::find_all();

?>
<?php include("includes/sidebar.php"); ?>
<?php include("includes/content-top.php"); ?>
<?php include("includes/content.php"); ?>
<?php include("includes/footer.php"); ?>
```

In content.php kunnen we nu deze variabelen aanspreken:

```php
                    <div class="col mr-2">
                        <div class="text-xs font-weight-bold text-primary text-uppercase mb-1">Users</div>
                        <div class="h5 mb-0 font-weight-bold text-gray-800"><?php echo count($aantalUsers);?></div>
                    </div>
                    <div class="col-auto">
                        <i class="fas fa-calendar fa-2x text-gray-300"></i>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <!-- Earnings (Monthly) Card Example -->
    <div class="col-12 col-md-4 mb-4">
        <div class="card border-left-success shadow h-100 py-2">
            <div class="card-body">
                <div class="row no-gutters align-items-center">
                    <div class="col mr-2">
                        <div class="text-xs font-weight-bold text-success text-uppercase mb-1">Photos</div>
                        <div class="h5 mb-0 font-weight-bold text-gray-800"><?php echo count($aantalPhotos);?></div>
                    </div>
                    <div class="col-auto">
                        <i class="fas fa-dollar-sign fa-2x text-gray-300"></i>
                    </div>
                </div>
            </div>
        </div>
    </div>


    <!-- Pending Requests Card Example -->
    <div class="col-12 col-md-4 mb-4 mb-4">
        <div class="card border-left-warning shadow h-100 py-2">
            <div class="card-body">
                <div class="row no-gutters align-items-center">
                    <div class="col mr-2">
                        <div class="text-xs font-weight-bold text-warning text-uppercase mb-1">Comments</div>
                        <div class="h5 mb-0 font-weight-bold text-gray-800"><?php echo count($aantalUsers);?></div>
                    </div>
```

Resultaat:
Surf naar admin/index.php



6.2. GOOGLE API CHARTS

Google naar **google charts** en kies daar **pie chart**.
https://developers.google.com/chart/interactive/docs/quick_start

Knip het script die we nodig hebben en plak deze in admin/includes/header.php.

```
<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
```

https://developers.google.com/chart/interactive/docs/gallery/piechart

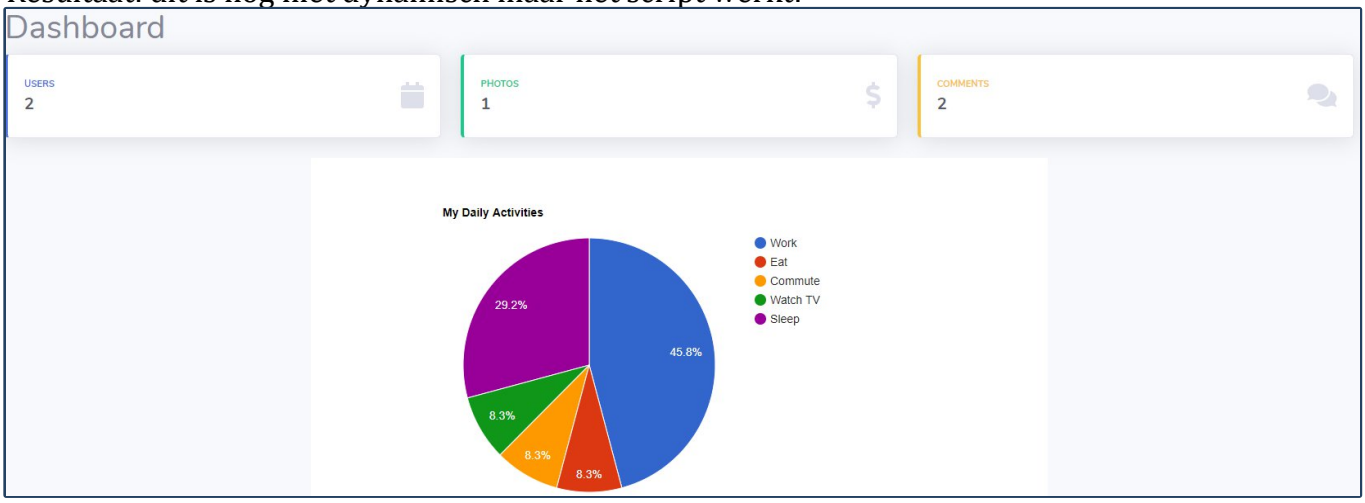
Plak nu enkel het script van de piechart onder het voorgaande script.

```
<!-- Bootstrap Core JavaScript -->
<script src="js/bootstrap.min.js"></script>

<script type="text/javascript">
    google.charts.load('current', {'packages':['corechart']});
    google.charts.setOnLoadCallback(drawChart);

    function drawChart() {

        var data = google.visualization.arrayToDataTable([
            ['Task', 'Hours per Day'],
            ['Work',     11],
            ['Eat',      2],
            ['Commute',  2],
            ['Watch TV', 2],
            ['Sleep',    7]
        ]);

        var options = {
            title: 'My Daily Activities'
        };

        var chart = new google.visualization.PieChart(document.getElementById('piechart'));

        chart.draw(data, options);
    }
</script>
```

Open terug content.php en pas aan als volgt:

```
        <!-- Pending Requests Card Example -->
        <div class="col-12 col-md-4 mb-4 mb-4">
            <div class="card border-left-warning shadow h-100 py-2">
                <div class="card-body">
                    <div class="row no-gutters align-items-center">
                        <div class="col mr-2">
                            <div class="text-xs font-weight-bold text-warning text-upperca
                            <div class="h5 mb-0 font-weight-bold text-gray-800"><?php echo
                        </div>
                        <div class="col-auto">
                            <i class="fas fa-comments fa-2x text-gray-300"></i>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <div class="row">
        <div class="mx-auto" id="piechart" style="width: 900px; height:500px;"></div>
    </div>
```

Resultaat: dit is nog niet dynamisch maar het script werkt!



Laten we work, eat, commute, watch tv en sleep vervangen door het volgende, nl.:
Aantal Views, comments, users en photos.

6.3. PAGE VIEWS

Elke keer we een pagina refreshen wordt er een sessie geladen. We zullen tellen hoeveel sessies er geladen worden. Dit noemen we PAGE VIEWS.

Open **blog/admin/includes/Session.php**:

```
/**automatisch starten van een sessie**/
function __construct(){
    session_start();
    $this->visitor_count();
    $this->check_the_login();
    $this->check_message();
}

public function visitor_count(){
    if(isset($_SESSION['count'])){
        return $this->count = $_SESSION['count']++;
    }else{
        return $_SESSION['count'] = 1;
    }
}
```

6.4. UNIVERSELE COUNT METHODE

We zullen een methode creëeren die het aantal records telt in onze database. Deze methode zullen we een aantal keren aanspreken binnen onze website, vandaar een universele count methode.

Open **Db_object.php** en pas als volgt aan:

```php
public function delete(){
    global $database;
    $sql = " DELETE FROM " . static::$db_table . " ";
    $sql .= " WHERE id= " . $database->escape_string($this->id);
    $sql .= " LIMIT 1";

    $database->query($sql);
    return (mysqli_affected_rows($database->connection) == 1) ? true : false;
}

public static function count_all()
{
    global $database;
    $sql = "SELECT COUNT(*) FROM " . static::$db_table;
    $result_set = $database->query($sql);
    $row = mysqli_fetch_array($result_set);

    return array_shift($row);
}
```

Open nu terug **content.php** en pas bovenstaande toe:
dit telt nu het aantal records in photo, comments en users opnieuw. Alles zal blijven werken.

```php
                    <div class="text-xs font-weight-bold text-primary text-uppercase mb-1">Users</div>
                    <div class="h5 mb-0 font-weight-bold text-gray-800"><?php echo User::count_all();?></div>
                </div>
                <div class="col-auto">
                    <i class="fas fa-calendar fa-2x text-gray-300"></i>
                </div>
            </div>
        </div>
    </div>
</div>

<!-- Earnings (Monthly) Card Example -->
<div class="col-12 col-md-4 mb-4">
    <div class="card border-left-success shadow h-100 py-2">
        <div class="card-body">
            <div class="row no-gutters align-items-center">
                <div class="col mr-2">
                    <div class="text-xs font-weight-bold text-success text-uppercase mb-1">Photos</div>
                    <div class="h5 mb-0 font-weight-bold text-gray-800"><?php echo Photo::count_all();?></div>
                </div>
                <div class="col-auto">
                    <i class="fas fa-dollar-sign fa-2x text-gray-300"></i>
                </div>
            </div>
        </div>
    </div>
</div>


<!-- Pending Requests Card Example -->
<div class="col-12 col-md-4 mb-4 mb-4">
    <div class="card border-left-warning shadow h-100 py-2">
        <div class="card-body">
            <div class="row no-gutters align-items-center">
                <div class="col mr-2">
                    <div class="text-xs font-weight-bold text-warning text-uppercase mb-1">Comments</div>
                    <div class="h5 mb-0 font-weight-bold text-gray-800"><?php echo Comment::count_all();?></div>
                </div>
                <div class="col-auto">
                    <i class="fas fa-comments fa-2x text-gray-300"></i>
                </div>
            </div>
        </div>
    </div>
</div>
```

Nu kunnen we ook onze pie chart aanpassen. Open **admin/includes/header.php** en pas aan als volgt:

```
var data = google.visualization.arrayToDataTable([
    ['Task', 'Hours per Day'],
    ['Views',     <?php echo $session->count; ?>],
    ['Comments',     <?php echo Comment::count_all(); ?>],
    ['Users',   <?php echo User::count_all(); ?>],
    ['Photos',  <?php echo Photo::count_all(); ?>]

]);

var options = {
    legend:'none',
    pieSliceText: 'label',
    title: 'My Daily Activities',
    backgroundColor: 'transparent'
};
```

7. FRONT-END: index.php

Hier gaan we onze photos toevoegen in de front-end van ons syteem, nl. in **blog/index.php** die ook de homepagina is.

Wanneer we dit bestand bekijken zien we dat hij verwijst naar blog/includes/**header.php.** Op dit niveau moeten we onze classes aanspreken om de photos te kunnen weergeven.

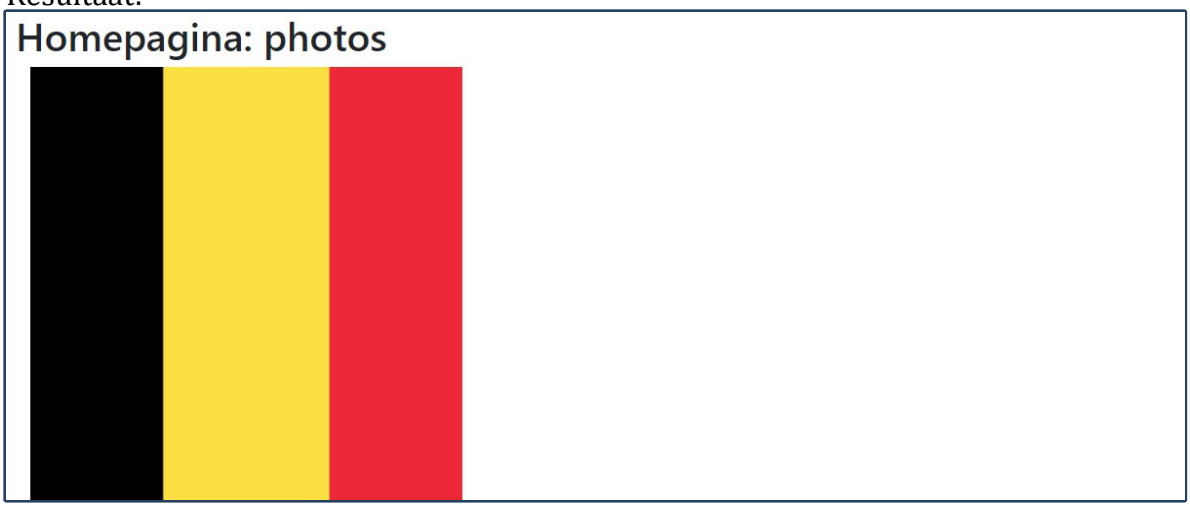
Open blog/includes/**header.php:**
via init hebben we toegang tot onze classes.

```php
<?php
include("includes/header.php");
$photos = Photo::find_all();
?>
```

Ga nu terug naar **blog/index.php**.

```php
<div class="container-fluid">
    <div class="row">
        <div class="col-12">
            <h1>Homepagina: photos</h1>
            <?php foreach($photos as $photo): ?>
            <div class="col-3">
                <a href="photo.php?id=<?php echo $photo->id; ?>">
                    <img class="img-fluid" src="<?php echo 'admin'.DS.$photo->picture_path(); ?>" alt="">
                </a>
            </div>
            <?php endforeach; ?>
        </div>
    </div>
</div>
<?php
include("includes/footer.php");
?>
```

Resultaat:



Wanneer je 3 photo's zou toevoegen dan zou je hierboven 3 verschillende photo's zien staan.
Wanneer we nu op één van de foto's van de blog klikken dan dient deze ons naar zijn detailpagina Photo.php te brengen.

Tot nu toe heb je een wisselwerking tussen front-end en back-end gezien. In principe zijn jullie nu zelf in staat om deze blog nog uit te breiden indien gewenst.

8. PAGINEREN

Zorg er eerst voor dat je een 20-tal photos in je database hebt opgeladen.
Hetgeen we wensen creëeren is bijvoorbeeld het volgende:
per pagina 10 items met paginering.

De url zou er als volgt kunnen uitzien:
localhost/blog/index.php?page=1
localhost/blog/index.php?page=2
enz...

Aangezien we object georiënteerd programmeren zullen we ook voor het pagineren een classe aanmaken:
In admin/includes maak **je paginate.php** aan:
We volgende logische redenering te volgen. Is er een: vorige pagina, aantal pagina's, er is nog een volgende pagina, er is geen vorige pagina meer.
Hiervoor zullen we de nodige methodes schrijven.
De functie ceil zorgt voor een afronding naar boven.

De offset zorgt voor de afbakening van het aantal items.
Bijvoorbeeld: wanneer je 10 items hebt per pagina, zal de offset ervoor zorgen dat de volgende 10 vanaf het 11de item worden getoond, ...

```php
<?php

class Paginate {

    public $current_page;
    public $items_per_page;
    public $items_total_count;

    public function __construct($page=1, $items_per_page=4, $items_total_count=0 ){
        $this->current_page = (int)$page;
        $this->items_per_page = (int)$items_per_page;
        $this->items_total_count = (int)$items_total_count;
    }

    public function next(){
        return $this->current_page + 1;
    }

    public function previous(){
        return $this->current_page - 1;
    }

    public function page_total(){
        return ceil( value: $this->items_total_count/$this->items_per_page);
    }

    public function has_previous(){
        return $this->previous() >= 1 ? true : false;
    }

    public function has_next(){
        return $this->next() <= $this->page_total() ? true : false;
    }

    public function offset() {
        return ($this->current_page -1 ) * $this->items_per_page;
    }
} // Paginate Class
?>
```

Vergeet de class niet toe te voegen in init.php:

```php
require_once(INCLUDES_PATH.DS."functions.php");
require_once(INCLUDES_PATH.DS."new_config.php");
require_once(INCLUDES_PATH.DS."database.php");
require_once(INCLUDES_PATH.DS."db_object.php");
require_once(INCLUDES_PATH.DS."user.php");
require_once(INCLUDES_PATH.DS."photo.php");
require_once(INCLUDES_PATH.DS."comment.php");
require_once(INCLUDES_PATH.DS."session.php");
require_once(INCLUDES_PATH.DS."paginate.php");
```

Open **blog/index.php** en pas als volgt aan:
Wanneer er een page id in de url staat dan halen we die er uit zodat we weten op welke pagina we ons bevinden op dat moment. In het andere geval tonen we altijd pagina 1.
De variabele items per page toont het aantal items die we wensen te zien per pagina.
De variabele items_total_count telt het aantal photos in de database.
De object $paginate variabele wordt gevuld met deze items. Die worden dan doorgegeven aan de class paginate.php voor verdere verwerking in de methodes van de class.
Het sql statement haalt vervolgens de variabelen uit de methodes van de class om de query per pagina te kunnen uitvoeren.

```php
<?php

    $page = !empty($_GET['page']) ? (int)$_GET['page'] : 1;
    $items_per_page = 4;
    $items_total_count = Photo::count_all();

    $paginate = new Paginate( $page, $items_per_page, $items_total_count);


    $sql = "SELECT * FROM photos ";
    $sql .= "LIMIT {$items_per_page} ";
    $sql .= "OFFSET {$paginate->offset()}";
    $photos = Photo::find_this_query($sql);


    //$photos = Photo::find_all();

?>
```

Probeer nu maar eens uit en surf naar: localhost/blog/index.php?page=1 of 2 of 3 ...
In ons voorbeeld worden er telkens 4 items per pagina getoond.

Nu dat de paginatie werkt, gaan we de navigatie links hiervoor maken binnen **blog/index.html**:
als eerste zullen we de vorige en volgende buttons creëeren met de classes uit bootstrap:

```
<?php endforeach; ?>
</div>
    <div class="row">
      <ul class="pager">
          <?php
          if($paginate->page_total() > 1) {
              if($paginate->has_next()) {
                  echo "<li class='next'><a href='index.php?page={$paginate->next()}'>Next</a></li>";
                }
              if($paginate->has_previous()) {
                  echo "<li class='previous'><a href='index.php?page={$paginate->previous()}'>Previous</a></li>";
              }
          }
          ?>
      </ul>
    </div>
```

De paginanummers zelf voegen we als volgt toe:
We gebruiken een loop die wordt uitgevoerd zolang het aantal pagina's kleiner of gelijk is aan de teller.

```
if($paginate->page_total() > 1) {
    if($paginate->has_next()) {
        echo "<li class='next'><a href='index.php?page={$paginate->next()}'>Next</a></li>";
      }

    for ($i=1; $i <= $paginate->page_total(); $i++) {
      if($i == $paginate->current_page) {
        |   echo  "<li class='active'><a href='index.php?page={$i}'>{$i}</a></li>";
      } else {
        |   echo  "<li><a href='index.php?page={$i}'>{$i}</a></li>";
      }
    }

    if($paginate->has_previous()) {
        echo "<li class='previous'><a href='index.php?page={$paginate->previous()}'>Previous</a></li>";
    }
}
?>
```

aangezien bootstrap zelf zijn javascript gebruikt werkt het verspringen van de active class al.
Wanneer je de actieve pagina dus wenst te stylen dan kun je dit eenvoudig in je css bestand aanpassen indien je wenst.

TIP: ipv de pager class kan je ook de pagination class gebruiken in je html.

9. HOSTING

Het online zetten van een applicatie vergt enkele aanpassingen. Er is namelijk een verschil tussen localhost en een domeinnaam met database koppeling.

STAPPENPLAN:
ONLINE IMPORTEREN DATABASE

- Exporteren database van de localhost (zie cursus database);
- Open dit bestand in een editor en zorg ervoor dat de databasenaam overeenkomt met de ONLINE DATABASENAAM die je krijgt van je hosting provider! Vervang deze dus ook hier!

```
--
-- Database: `gallery_db`
--
CREATE DATABASE IF NOT EXISTS `gallery_db` DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci;
USE `gallery_db`;
```

- IMPORTEER het SQL bestand in de database van je ONLINE provider.

AANPASSEN WEBSITE:
Laad de volledige folder op naar de root van je hosting. Let op: niet de blog folder zelf, maar wel alles die daar in zit!

config.PHP:
de user, paswoord en databasenaam dienen ook hier te worden aangepast. Deze 3 parameters krijg je van je hosting provider.
INIT.PHP:
Pas nu de paden aan in init.php:
De domeinnaam root benaderen we door de global $_SERVER variabele te gebruiken met de optie DOCUMENT_ROOT. Op deze locatie dient de startup pagina zich te bevinden van je site, nl. index.php.

```
defined('DS') ? null: define('DS', DIRECTORY_SEPARATOR);
define('SITE_ROOT', $_SERVER['DOCUMENT_ROOT']);
```

PHP.INI (optioneel)
Het is afhankelijk van de hosting provider of je een eigen **php.ini** bestand in je root mag zetten. Een php.ini bestand bevat settings eigen aan de php server zelf. Zo kan je bijvoorbeeld errormeldingen aanzetten (enable). Dit is zeer interessant wanneer je een site online zet. De errors begeleiden je namelijk in het oplossen van die problemen.
Indien je geen toegang hebt, dien je aan je provider te vragen hoe je deze errors kan aan- en uitzetten.
In php.ini doe je dit met de volgende code:
error_reporting = E_ALL
display_errors = On

10.EXTRA UITLEG OVER =FILES

10.1. FILE TOEGANG

Wanneer we bestanden gaan uploaden naar mappen op de webserver dan dienen we deze in te stellen.

WINDOWS

Voor windows gebruikers:

- rechts klik op het bestand, properties, beveiliging, wijzig
- probleem die je kan hebben:
    - afhankelijk van de versie van windows kan file toegang anders ingesteld staan.

MAC, LINUX

READ = user (ja), groep (ja), ander (ja) = 4
WRITE = user (ja), groep (ja), ander (nee) = 2
E**X**ECUTE = user(ja), groep(ja), ander(nee) = 1

Bijv. rwx = 777

Toegang commando's

sudo = run als een superuser
chmod = verander toegang
chown = verander eigenaar

10.2. FILE UPLOAD

File upload hangt af van de server of hosting waar je je bestanden zal uploaden.
Zorg er steeds voor dat je hosting hebt waar je het **php.ini** bestand kan aanpassen!
Wat kan je in dit bestand wijzigen: **localhost => phpinfo();**

Standaarden anno 2018:
file_uploads = True, 1 of On (afhankelijk van de soort server)
post_max_size = 8M
upload_max_file_size = 2M
memory_limit = 128M
max_input_time = -1 - no limit
upload_tmp_dir = NULL
max_execution_time = 30 seconden

Bovenstaande zijn standaarden. Deze standaarden kunnen echter gewijzigd worden in **php.ini.** Dit bestand kunnen we vinden onder de versie van php die we op dit moment draaien op de server.

10.2.1. FILE STRUCTUUR VOOR UPLOAD

Hiervoor maken we gebruik van de Super Global $_FILES['mijnbestand'].
De upload zal gebeuren met een associatieve array

k= key
v= value

(k) name => (v) bestandsnaam
(k) type => (v) jpg, png, gif, doc, txt
(k) grootte => (v) bytes
(k) tmp_name => temporary name
(k) error => de error code

Bovenstaande zijn de sleutels en waarden die we tijdens een file upload kunnen aanspreken.

Voorbeeld:
$size = $_FILES[bestandsnaam]['grootte'];