

PHP OOP van nul naar job-ready (junior)

Bouw je eigen CMS – MiniCMS Pro

1. Voorwoord

Waarom dit boek bestaat

De meeste PHP-cursussen doen één van deze dingen fout:

- ze blijven **procedureel** en “plakken” OOP erop
- ze starten **abstract** (class dit, object dat) zonder zichtbaar resultaat
- ze gebruiken **frameworks** waardoor je niet leert *waarom* iets werkt
- ze slaan stappen over “ omdat dat later wel duidelijk wordt”

Dit boek doet het tegenovergestelde.

Je bouwt **één echte applicatie**:

☞ **MiniCMS Pro**, een professioneel CMS met dashboard, login, rollen, CRUD, database, MVC en security.

Alles wat je leert heeft **een zichtbare bestemming**.

Hoe je dit boek gebruikt (belangrijk)

- **Volg chronologisch.** Sla niets over.
- **Typ mee**, ook al lijkt het triviaal.
- **Run de code na elke les.**
- Zie je een fout? Dat is geen probleem maar leerstof.
- Vergelijk **altijd** met procedurele PHP wanneer dat gevraagd wordt.

Dit is geen referentieboek. Dit is een **opleidingstraject**.

MODULE 0 — Setup & PHP Mental Model

Doel van deze module

Na deze module:

- begrijp je **hoe PHP requests werken**
- weet je **waarom errors soms “verdwijnen”**
- kan je **debuggen zonder frameworks**
- heb je een **correcte lokale setup**
- heb je een **mentaal model** dat later OOP en MVC logisch maakt

LES 0.1 — Lokale ontwikkelomgeving opzetten

A) Doel

Een werkende PHP 8.x omgeving met zichtbare fouten.

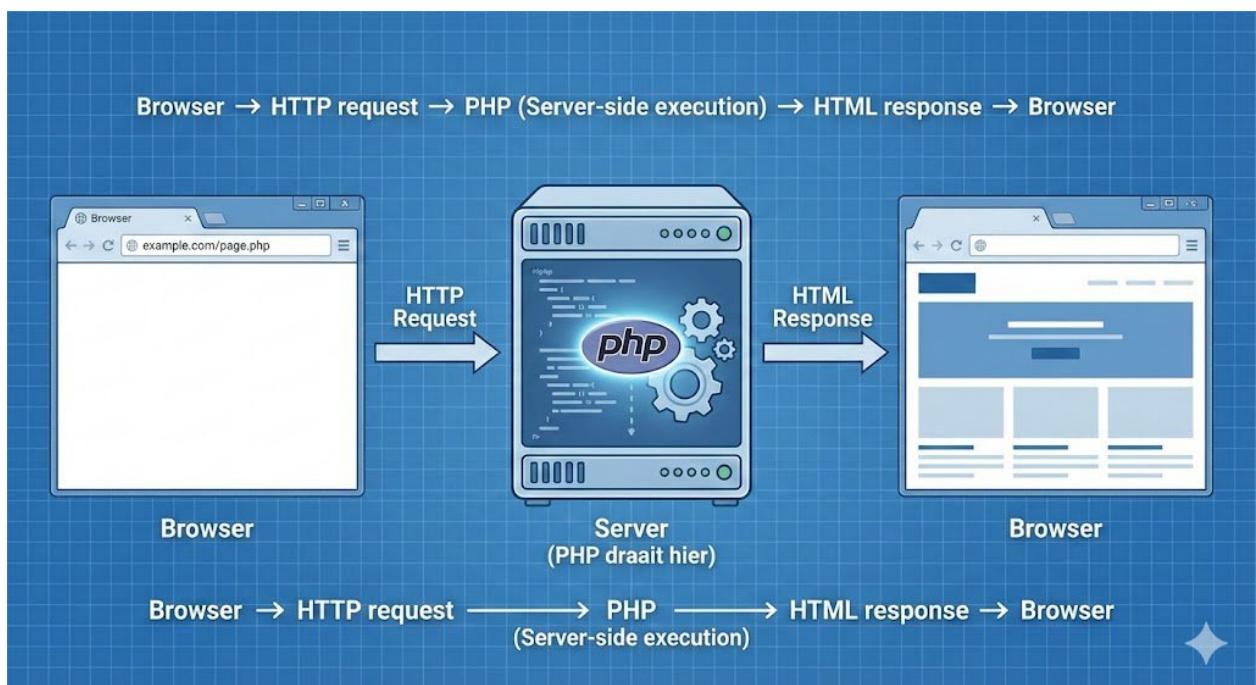
B) Nieuwe termen

Local development environment Een lokale server op je eigen computer om PHP te draaien zonder internet.

C) Concept

PHP draait **niet** in de browser. PHP draait op een **server** die HTML terugstuurt.

Browser → HTTP request → PHP → HTML response → Browser



D) Click-per-click stappen

Stap 1: Installeer een local server

Kies **één** van deze opties:

- XAMPP
- WAMP
- Laragon (aanbevolen)

Installeer met **PHP 8.x** geselecteerd en start je server.

Stap 2: Projectmap aanmaken

Ga naar je webroot:

- XAMPP: C:\xampp\htdocs
- WAMP: C:\wamp64\www
- Laragon: C:\laragon\www

Maak een map:

minicms-pro

Stap 3: Eerste PHP bestand

Maak bestand:

C:\laragon\www\minicms-pro\index.php

Inhoud:

```
<?php  
echo 'MinicMS Pro is alive';
```

echo is een taalconstructie in PHP die tekst uitstuurt naar de browser.

Concreet:

- PHP voert code uit op de server
- echo zegt tegen PHP:
 ↳ “stuur deze tekst mee in de HTML response”

E) Browser test

Open Chrome en ga naar:

<http://localhost/minicms-pro>

Verwachte output:

MinicMS Pro is alive

F) Checkpoint test

? Zie je tekst in de browser ✓ Ja → OK ✗ Nee → server niet actief of verkeerde map

G) Oefening

Opgave Verander de tekst zodat je eigen naam zichtbaar is.

Oplossing

```
<?php  
echo 'MiniCMS Pro by Tom';
```

H) Veelgemaakte fouten

- .php openen via dubbelklik → werkt NIET
- PHP code in .html → wordt niet uitgevoerd

I) Mini-quiz

1. Draait PHP in de browser?
2. Wat stuurt PHP terug?
3. Waar moet je PHP bestanden plaatsen?

Antwoorden

1. Nee
2. HTML
3. In de webroot van je server

J) Wat je nu kan

- PHP lokaal draaien
- Files correct plaatsen
- Basis request testen

K) Samenvatting

- PHP draait server-side
- Browser ziet alleen HTML
- Local server is verplicht
- Bestanden in webroot
- Test altijd via localhost

LES 0.2 — PHP error reporting begrijpen

A) Doel

Nooit meer “white screen of death”.

De **White Screen of Death** betekent:

Je opent een pagina in de browser en ziet **een volledig wit scherm, zonder foutmelding, zonder tekst, zonder uitleg**.

De pagina lijkt “kapot”, maar je weet **niet waarom**.

Waarom gebeurt dit?

In PHP gebeurt dit bijna altijd door **een fout in de code**, maar:

- PHP **verbergt fouten standaard**
- zeker in productie-achtige instellingen
- de browser krijgt dan **geen output**

Resultaat:

☞ **wit scherm**

B) Nieuwe termen

Error reporting Instellingen die bepalen welke fouten PHP toont.

C) Concept

PHP **verbergt fouten standaard** in productie. Tijdens ontwikkeling wil je **alles zien**.

D) Click-per-click

Bovenaan index.php:

```
<?php  
declare(strict_types=1);  
  
error_reporting(E_ALL);  
ini_set('display_errors', '1');
```

Deze regels zorgen ervoor dat PHP zich streng, eerlijk en transparant gedraagt tijdens ontwikkeling.

We leggen ze één voor één uit.

```
declare(strict_types=1);
```

Wat is declare?

declare is een **instructie aan de PHP-engine**
die bepaalt **hoe PHP zich moet gedragen** in dit bestand.

Het is **geen functie en geen variabele**.

Wat betekent strict_types=1?

Het betekent:

“PHP, wees **strengh** met types.”

Zonder strict_types probeert PHP “behulpzaam” te zijn door types automatisch om te zetten.

Voorbeeld zonder strict typing

```
function telOp(int $a, int $b) {  
    return $a + $b;  
}  
  
echo telOp("5", "3");  
  
☞ PHP zet "5" automatisch om naar 5.  
☞ Geen fout.
```

Met strict_types=1

```
declare(strict_types=1);  
  
function telOp(int $a, int $b) {  
    return $a + $b;  
}  
  
echo telOp("5", "3");
```

✗ Fatal error

☞ Types kloppen niet.

Waarom gebruiken wij strict_types=1?

- Fouten worden **vroeger** zichtbaar
- Minder verborgen bugs
- Code is voorspelbaar
- Dit is **professionele PHP 8.x stijl**

! We gebruiken dit **altijd** in deze cursus.

```
error_reporting(E_ALL);
```

Wat is error reporting?

Error reporting bepaalt **welke soorten fouten PHP mag melden**.

E_ALL betekent letterlijk:

“Rapporteer **alle** fouten, waarschuwingen en notices.”

Waarom is dit belangrijk?

Zonder E_ALL zie je bijvoorbeeld:

- notices niet
- warnings niet
- deprecated meldingen niet

☞ Die fouten **zijn er wel**, maar je ziet ze niet.

Concreet voorbeeld

```
echo $onbestaandeVariabele;
```

Met E_ALL:

- duidelijke warning
- lijnnummer

Zonder E_ALL:

- mogelijk niets
- of wit scherm

```
ini_set('display_errors', '1');
```

Wat doet ini_set?

ini_set wijzigt **PHP-instellingen tijdens runtime**
(dus terwijl het script draait).

Wat betekent 'display_errors'?

Dit is een PHP-instelling die bepaalt:

“Mogen fouten zichtbaar zijn in de browser?”

Wat betekent '1'?

- '1' = aan
- '0' = uit

Waarom combineren we dit met error_reporting(E_ALL)?

Instelling	Wat gebeurt
Alleen E_ALL	PHP registreert fouten
Alleen display_errors=1	PHP toont fouten die gemeld worden
Beide samen	PHP meldt én toont fouten

☞ Samen voorkomen ze de **white screen of death**.

Zeer belangrijk: wanneer gebruiken we dit WEL en NIET

Tijdens ontwikkeling (zoals in deze cursus)

JA

```
error_reporting(E_ALL);
```

```
ini_set('display_errors', '1');
```

In productie

NEE

In productie:

- fouten loggen
- niet tonen aan bezoekers

Samenvatting

- declare(strict_types=1) → strenge types

- error_reporting(E_ALL) → alle fouten melden
- ini_set('display_errors', '1') → fouten tonen
- Samen voorkomen ze witte schermen
- Standaard in professionele PHP

E) Test met fout

Onder je echo:

```
echo $onbestaandeVariabele;
```

Verwachte output: Notice / Warning in browser

F) Waarom dit belangrijk is

Zonder errors:

- denk je dat PHP “niet werkt”
- debug je blind
- leer je niets

G) Oefening

Maak een syntax error en los hem op.

H) Veelgemaakte fouten

- Error reporting pas later toevoegen
- Denken dat een wit scherm “normaal” is

I) Mini-quiz

1. Waarom toont productie geen errors?
2. Wat betekent E_ALL?

Antwoorden

1. Security
2. Alle fouten en waarschuwingen

J) Wat je nu kan

- Errors zichtbaar maken
- Fouten herkennen
- Debuggen starten

K) Samenvatting

- Errors zijn leerstof
- White screen is fout
- Toon alles lokaal
- Verberg in productie
- Debug zichtbaar

LES 0.3 — Request & Response mental model

A) Doel

Begrijpen **wat PHP exact doet** per page load.

B) Nieuwe termen

HTTP request Een aanvraag van de browser.

HTTP response Het antwoord van de server.



C) Concept (cruciaal)

Elke page refresh:

1. Browser vraagt pagina
2. PHP script start van boven
3. PHP voert code uit
4. PHP stopt
5. HTML wordt gestuurd

! PHP “onthoudt” niets tussen requests (dit verandert later met sessions)

D) Visualisatie

```
<?php  
echo 'Start';  
echo ' - ';  
echo 'Einde';
```

Elke refresh = alles opnieuw.

E) Oefening

Refresh 5 keer. Vraag jezelf af: wat blijft bestaan?

Antwoord: niets.

F) Veelgemaakte misconception

“Mijn variabele bestaat nog van daarnet”

✗ FOUT PHP start altijd opnieuw.

G) Mini-quiz

1. Wanneer stopt PHP?
2. Wat blijft bestaan zonder sessions?

Antwoorden

1. Na response
2. Niets

K) Samenvatting

- PHP is stateless
- Elke request start opnieuw
- Geen geheugen tussen loads
- Sessions lossen dit op
- MVC bouwt hierop voort

MODULE 1 — Static Admin Dashboard (Tailwind)

Dashboard-first aanpak. Eerst zien, dan structureren, dan abstraheren.

Waarom deze module zo belangrijk is

Voor we ook maar één class schrijven, bouwen we **een volledig zichtbaar admin dashboard**:

- professionele layout
- echte UI-componenten
- echte data-visualisatie (ApexCharts)
- opgesplitst met include en require

Dit is **bewust nog GEEN MVC(=Model View Controller)**. Dat verschil gaan we expliciet benoemen en later oplossen.

MODULE 1 – Overzicht

Wat we bouwen in deze module

- Admin dashboard (/admin)
- Sidebar + topbar
- Content cards
- Tabel (posts overzicht)
- Form (dummy)
- ApexCharts grafieken
- Opsplitsing via include

LES 1.1 — Admin folderstructuur & entry point

A) Doel

- Een **aparte admin zone**
- Eén duidelijk startpunt: admin/index.php
- Basisstructuur die later MVC aankan

B) Nieuwe termen

Admin dashboard

Een afgeschermde beheerdersomgeving voor content en instellingen.

Entry point

Het eerste PHP-bestand dat een request afhandelt.

C) Concept (vergelijking met procedureel PHP)

Procedureel (klassiek):

dashboard.php
posts.php
users.php

Problemen:

- geen centrale layout
- copy-paste
- moeilijk uitbreidbaar

Onze aanpak:

- 1 entry point
- content wisselt
- layout blijft vast

D) Click-per-click stappen

Stap 1: Maak admin structuur

In je projectroot:

```
/minicms-pro
  /admin
    /includes
    /pages
  index.php
```

Stap 2: admin/index.php

```
admin/index.php
```

```
<?php
declare(strict_types=1);

error_reporting(E_ALL);
ini_set('display_errors', '1');

echo 'Admin dashboard entry point';
```

Stap 3: Test in browser

<http://localhost/minicms-pro/admin>

Verwachte output

Admin dashboard entry point

E) Checkpoint test

✓ Admin map laadt ✓ PHP errors zichtbaar ✓ Geen white screen

F) Veelgemaakte fouten

- admin.php gebruiken i.p.v. map
- admin mengen met frontend
- geen aparte entry point

G) Mini-quiz

1. Waarom aparte admin map?
2. Waarom één index.php?

Antwoorden

1. Security en structuur
2. Centrale controle

H) Wat je nu kan

- Admin zone structureren
- Entry point begrijpen

I) Samenvatting

- Admin is gescheiden
- Eén entry point
- Layout volgt later
- Geen MVC nog
- Basis is correct

LES 1.2 — Tailwind CSS via CDN (dashboard-first)

A) Doel

- Tailwind gebruiken **zonder build**
- Focus op layout, niet tooling

B) Nieuwe termen

Tailwind CSS

Utility-first CSS framework met kleine herbruikbare classes.

CDN

Extern gehoste bestanden die je direct kan laden.

C) Waarom CDN eerst

- Geen Node
- Geen build
- Sneller leren
- Later refactoren we dit

D) Click-per-click

Stap 1: Basis HTML structuur

admin/index.php

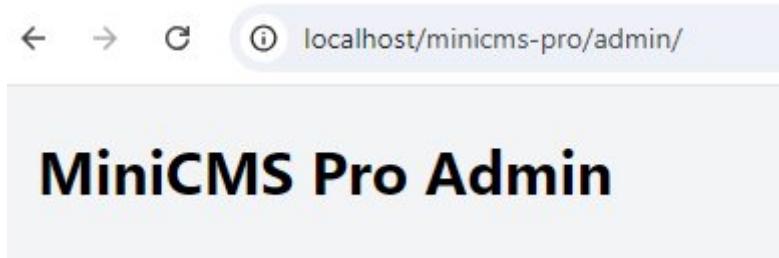
```
<?php
declare(strict_types=1);
error_reporting(E_ALL);
ini_set('display_errors', '1');
?>
<!DOCTYPE html>
<html lang="nl">
<head>
```

```

<meta charset="UTF-8">
<title>MiniCMS Pro – Admin</title>
<script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-gray-100">
  <h1 class="text-3xl font-bold p-6">
    MiniCMS Pro Admin
  </h1>
</body>
</html>

```

Stap 2: Browser test



Je ziet:

- grijze achtergrond
 - grote titel
- Tailwind werkt

E) Veelgemaakte fouten

- CDN vergeten
- verkeerde body class
- CSS file proberen includen

F) Mini-quiz

1. Waarom geen build?
2. Wat doet bg-gray-100?

Antwoorden

1. Focus op layout
2. Achtergrondkleur

G) Wat je nu kan

- Tailwind laden
- Utility classes gebruiken

H) Samenvatting

- CDN is tijdelijk
- Tailwind is zichtbaar
- Geen tooling ruis
- Layout komt eerst
- Refactor later

LES 1.3 — Dashboard layout: sidebar + topbar

A) Doel

- Professionele dashboardstructuur
- Vast patroon dat je overal ziet

B) Nieuwe termen

Sidebar

Verticale navigatie aan de zijkant.

Topbar

Horizontale balk bovenaan met acties.

C) Concept (mentaal model)

Dashboard = **flex container**

- links: sidebar (fixed width)
- rechts: content (flex-grow)

D) Click-per-click

admin/index.php (volledig vervangen)

```
<?php
declare(strict_types=1);
error_reporting(E_ALL);
ini_set('display_errors', '1');
?>
<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <title>MiniCMS Pro – Admin</title>
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-gray-100">

<div class="flex min-h-screen">

    <!-- Sidebar -->
    <aside class="w-64 bg-gray-900 text-white p-6">
        <h2 class="text-xl font-bold mb-6">MiniCMS Pro</h2>
        <nav class="space-y-3">
            <a href="#" class="block text-gray-300 hover:text-white">Dashboard</a>
            <a href="#" class="block text-gray-300 hover:text-white">Posts</a>
            <a href="#" class="block text-gray-300 hover:text-white">Users</a>
        </nav>
    </aside>
```

```

<!-- Main -->
<main class="flex-1">
    <!-- Topbar -->
    <header class="bg-white shadow px-6 py-4 flex justify-between">
        <span class="font-semibold">Dashboard</span>
        <span class="text-sm text-gray-600">Admin</span>
    </header>

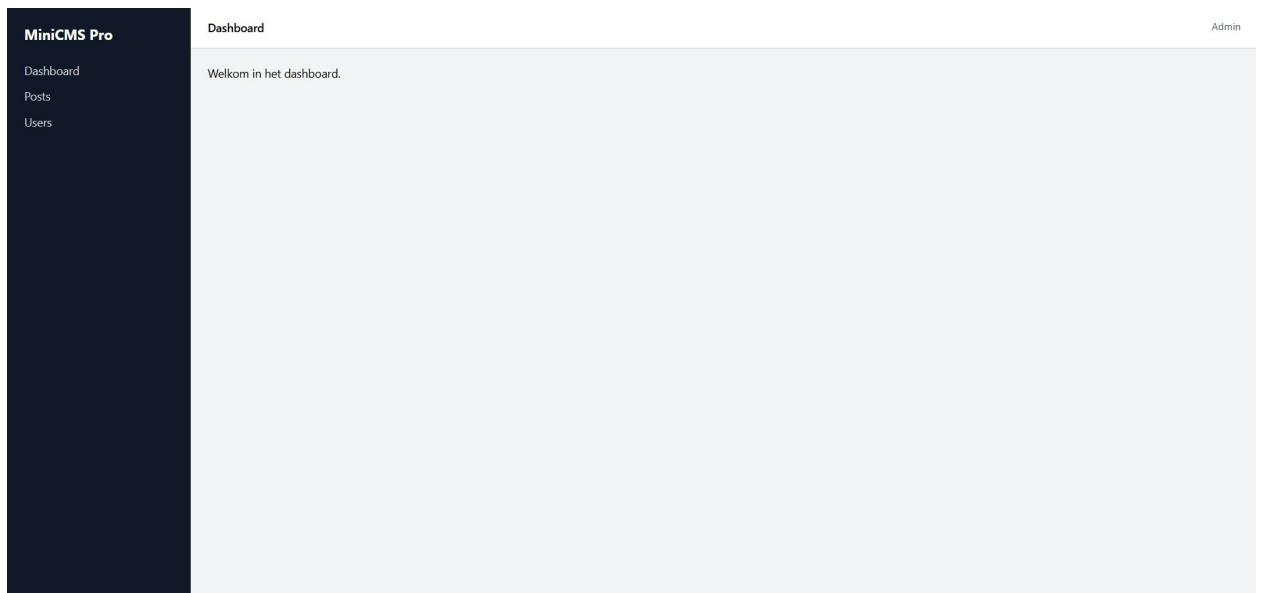
    <!-- Content -->
    <section class="p-6">
        <p>Welkom in het dashboard.</p>
    </section>
</main>

</div>

</body>
</html>

```

E) Verwachte output



- Donkere sidebar links
- Witte topbar
- Content rechts

F) Veelgemaakte fouten

- flex vergeten
- sidebar niet fixed
- padding vergeten

G) Mini-quiz

1. Waarom `flex-1`?
2. Waarom `min-h-screen`?

Antwoorden

1. Neemt resterende ruimte
2. Volledige hoogte

H) Wat je nu kan

- Dashboard layout bouwen
- Sidebar/topbar begrijpen

I) Samenvatting

- Dashboard = flex
- Sidebar vast
- Content flexibel
- Tailwind is voorspelbaar
- Basis staat

LES 1.4 — Cards & tabellen (zichtbare CMS data)

A) Doel

- Visuele “data”
- Voorbereiding op echte backend

B) Concept

We tonen **fake data**, maar echte UI.

Later:

- cards → database
- tabel → CRUD
- cijfers → queries

C) Click-per-click

Voeg toe in `<section class="p-6">`

```
<div class="grid grid-cols-3 gap-6 mb-8 m-4">
  <div class="bg-white p-6 rounded shadow">
    <p class="text-gray-500 text-sm">Posts</p>
    <p class="text-2xl font-bold">12</p>
  </div>
  <div class="bg-white p-6 rounded shadow">
    <p class="text-gray-500 text-sm">Users</p>
    <p class="text-2xl font-bold">3</p>
  </div>
  <div class="bg-white p-6 rounded shadow">
    <p class="text-gray-500 text-sm">Views</p>
    <p class="text-2xl font-bold">1.245</p>
  </div>
</div>
```

D) Tabel toevoegen

```
<div class="bg-white rounded shadow p-6 m-4">
  <h3 class="font-bold mb-4">Laatste posts</h3>
  <table class="w-full text-sm">
    <thead>
      <tr class="text-left border-b">
        <th class="py-2">Titel</th>
        <th>Datum</th>
        <th>Status</th>
      </tr>
    </thead>
    <tbody>
      <tr class="border-b">
        <td class="py-2">Welkom</td>
        <td>2026-01-01</td>
        <td>Published</td>
      </tr>
      <tr>
        <td class="py-2">Tweede post</td>
        <td>2026-01-05</td>
        <td>Draft</td>
      </tr>
    </tbody>
  </table>
</div>
```

E) Wat je ziet

The screenshot shows the MiniCMS Pro dashboard interface. On the left is a dark sidebar with the title "MiniCMS Pro" and links for "Dashboard", "Posts", and "Users". The main content area has a header "Dashboard" and a sub-header "Welkom in het dashboard.". It features three cards: "Posts" (12), "Users" (3), and "Views" (1.245). Below these is a section titled "Laatste posts" with a table:

Titel	Datum	Status
Welkom	2026-01-01	Published
Tweede post	2026-01-05	Draft

- 3 cards
- 1 tabel
- Volledig dashboard-gevoel

F) Samenvatting

- UI eerst
- Data later

- CMS begint zichtbaar
- OOP krijgt straks context
- Motivatie ↑

LES 1.5 — ApexCharts integratie (dummy analytics)

A) Doel

- Echte grafieken
- Voorbereiding op database data

B) Nieuwe term

ApexCharts

JavaScript chart library voor dashboards.

C) Waarom nu al charts

- Dashboards hebben analytics
- Later vervangen we dummy data
- Geen redesign nodig

D) Click-per-click

Stap 1: CDN toevoegen

In <head>:

```
<script src="https://cdn.jsdelivr.net/npm/apexcharts"></script>
```

Stap 2: Chart container

Onder cards:

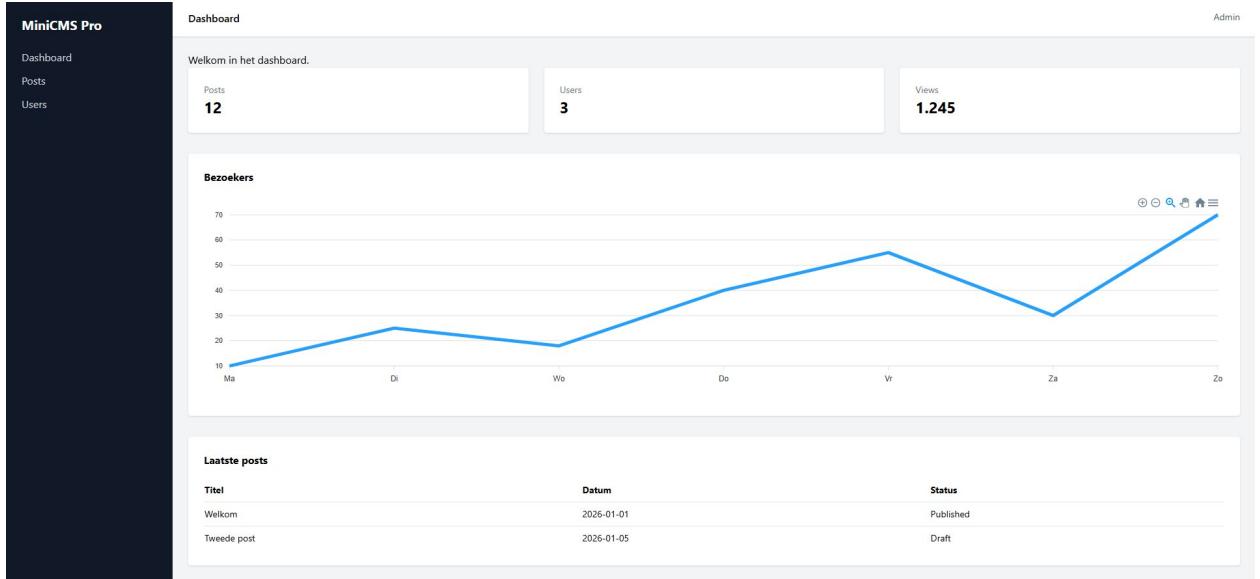
```
<div class="bg-white p-6 rounded shadow mb-8 m-4">
  <h3 class="font-bold mb-4">Bezoekers</h3>
  <div id="chart"></div>
</div>
```

Stap 3: Script onderaan body

```
<script>
var options = {
  chart: {
    type: 'line',
    height: 300
  },
  series: [
    {
      name: 'Bezoekers',
      data: [10, 25, 18, 40, 55, 30, 70]
    },
    xaxis: {
      categories: ['Ma', 'Di', 'Wo', 'Do', 'Vr', 'Za', 'Zo']
    }
};
```

```
var chart = new ApexCharts(document.querySelector("#chart"), options);
chart.render();
</script>
```

E) Verwachte output



- Lijngrafiek
- Volledig dashboard-gevoel

F) Belangrijke opmerking

! Deze data komt **NIET uit PHP !** Dit is bewust ! Later injecteren we PHP → JS

LES 1.6 — Dashboard opsplitsen met `include` en `require`

Dit is een **bewuste tussenstap**. Niet “de juiste architectuur”, wel **didactisch noodzakelijk**.

A) Doel

- Het dashboard **leesbaar en onderhoudbaar** maken
- Copy-paste elimineren
- Begrijpen **waarom dit nog geen MVC is**
- Voorbereiding op refactoring

B) Nieuwe termen

`include`

Laadt een PHP-bestand in op de plaats waar het wordt aangeroepen. Bij fout: **script gaat verder**.

`require`

Laadt een PHP-bestand in. Bij fout: **script stopt**.

Layout fragment

Een herbruikbaar stuk HTML/PHP, zoals header, sidebar of footer.

C) Concept (vergelijking met procedurele PHP)

Zonder includes (klassiek fout patroon)

```
<!-- dashboard.php -->
<header>...</header>
<aside>...</aside>
<section>...</section>

<!-- posts.php -->
<header>...</header>
<aside>...</aside>
<section>...</section>
```

Problemen:

- copy-paste
- wijziging op 10 plaatsen
- fouten sluipen binnen

Met includes (onze stap nu)

```
include header
include sidebar
include content
include footer
```

Voordelen:

- één bron per layoutstuk
- sneller aanpassen
- **leesbaar**

! Maar: *geen routing geen controllers geen models*

D) Click-per-click stappen

Stap 1: Maak includes structuur

```
/admin
  /includes
    header.php
    sidebar.php
    topbar.php
    footer.php
/pages
  dashboard.php
  index.php
```

Stap 2: admin/includes/header.php

```
admin/includes/header.php

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

```

<head>
    <meta charset="UTF-8">
    <title>MiniCMS Pro – Admin</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <script src="https://cdn.jsdelivr.net/npm/apexcharts"></script>
</head>
<body class="bg-gray-100">
```

<div class="flex min-h-screen">

Step 3: admin/includes/sidebar.php

admin/includes/sidebar.php

```

<aside class="w-64 bg-gray-900 text-white p-6">
    <h2 class="text-xl font-bold mb-6">MiniCMS Pro</h2>
    <nav class="space-y-3">
        <a href="#" class="block text-gray-300 hover:text-white">Dashboard</a>
        <a href="#" class="block text-gray-300 hover:text-white">Posts</a>
        <a href="#" class="block text-gray-300 hover:text-white">Users</a>
    </nav>
</aside>
```

Step 4: admin/includes/topbar.php

admin/includes/topbar.php

```

<header class="bg-white shadow px-6 py-4 flex justify-between">
    <span class="font-semibold">Dashboard</span>
    <span class="text-sm text-gray-600">Admin</span>
</header>
```

Step 5: admin/includes/footer.php

admin/includes/footer.php

```

</div> <!-- end flex -->

<script>
var options = {
    chart: {
        type: 'line',
        height: 300
    },
    series: [
        name: 'Bezoekers',
        data: [10, 25, 18, 40, 55, 30, 70]
    ],
    xaxis: {
        categories: ['Ma', 'Di', 'Wo', 'Do', 'Vr', 'Za', 'Zo']
    }
};

var chart = new ApexCharts(
    document.querySelector("#chart"),
    options
```

```
});  
chart.render();  
</script>
```

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

Stap 6: admin/pages/dashboard.php

```
admin/pages/dashboard.php
```

```
<section class="p-6">  
  
<div class="grid grid-cols-3 gap-6 mb-8">  
    <div class="bg-white p-6 rounded shadow">  
        <p class="text-gray-500 text-sm">Posts</p>  
        <p class="text-2xl font-bold">12</p>  
    </div>  
    <div class="bg-white p-6 rounded shadow">  
        <p class="text-gray-500 text-sm">Users</p>  
        <p class="text-2xl font-bold">3</p>  
    </div>  
    <div class="bg-white p-6 rounded shadow">  
        <p class="text-gray-500 text-sm">Views</p>  
        <p class="text-2xl font-bold">1.245</p>  
    </div>  
</div>  
  
<div class="bg-white p-6 rounded shadow mb-8">  
    <h3 class="font-bold mb-4">Bezoekers</h3>  
    <div id="chart"></div>  
</div>  
  
<div class="bg-white rounded shadow p-6">  
    <h3 class="font-bold mb-4">Laatste posts</h3>  
    <table class="w-full text-sm">  
        <thead>  
            <tr class="text-left border-b">  
                <th class="py-2">Titel</th>  
                <th>Datum</th>  
                <th>Status</th>  
            </tr>  
        </thead>  
        <tbody>  
            <tr class="border-b">  
                <td class="py-2">Welkom</td>  
                <td>2026-01-01</td>  
                <td>Published</td>  
            </tr>  
            <tr>  
                <td class="py-2">Tweede post</td>  
                <td>2026-01-05</td>  
                <td>Draft</td>  
            </tr>  
        </tbody>  
</table>
```

```

</div>

</section>

Stap 7: admin/index.php (entry point)

admin/index.php

<?php
declare(strict_types=1);

error_reporting(E_ALL);
ini_set('display_errors', '1');//toon fouten effectie in browser
require __DIR__ . '/includes/header.php';
require __DIR__ . '/includes/sidebar.php';
?>

<main class="flex-1">
    <?php require __DIR__ . '/includes/topbar.php'; ?>
    <?php require __DIR__ . '/pages/dashboard.php'; ?>
</main>

<?php
require __DIR__ . '/includes/footer.php';

```

E) Verwachte output

- Exact hetzelfde dashboard
- Geen duplicatie
- Overzichtelijke structuur

✓ Als dit werkt, ben je correct

F) Waarom __DIR__ gebruiken (zeer belangrijk)

Zonder __DIR__

```

require 'includes/header.php';

```

Probleem:

- werkt soms
- faalt bij subfolders
- breekt later routing

Met __DIR__

```

require __DIR__ . '/includes/header.php';

```

✓ Absoluut pad ✓ Altijd correct ✓ Professioneel

__DIR__ is een **magic constant** in PHP.

Het betekent:

Het absolute pad naar de map van het huidige PHP-bestand.

Voorbeeld:

Stel je bestand staat hier:

C:\wamp64\www\minicms-pro\admin\index.php

Dan is in dat bestand:

echo __DIR__;

Output:

C:\wamp64\www\minicms-pro\admin

Dus **zonder bestandsnaam**, alleen de map.

G) Veelgemaakte fouten + fixes

Fout	Oorzaak	Fix
Include niet gevonden	Relatief pad	Gebruik __DIR__
Lege pagina	require faalt	Check bestandsnaam
HTML tags verkeerd	Header/footer fout	Controleer opening/sluiting

H) Oefening

A) Opgave

Maak een tweede pagina posts.php in /pages met een simpele titel.

B) Requirements

- Zelfde layout
- Andere content
- Geen duplicatie

C) Oplossing

admin/pages/posts.php

The screenshot shows the MiniCMS Pro admin dashboard. On the left, there is a dark sidebar with the title 'MinicMS Pro' and three menu items: 'Dashboard', 'Posts', and 'Users'. The main content area has a header 'Dashboard' at the top. Below it, there is a section titled 'Posts beheer' with the sub-instruction 'Hier beheer je alle posts.' The right side of the screen has a light gray background with some faint text or icons that are mostly illegible due to the low resolution of the screenshot.

```

<section class="p-6">
    <h1 class="text-2xl font-bold">Posts beheer</h1>
    <p class="text-gray-600 mt-2">Hier beheer je alle posts.</p>
</section>

```

Voeg in index.php tijdelijk toe onder dashboard::

```
require __DIR__ . '/pages/posts.php';
```

The screenshot shows a web application interface. At the top, there is a header with the text "Laatste posts". Below the header is a table with three columns: "Titel", "Datum", and "Status". The table contains two rows of data:

Titel	Datum	Status
Welkom	2026-01-01	Published
Tweede post	2026-01-05	Draft

Below the table, there is a section titled "Posts beheer" with the sub-instruction "Hier beheer je alle posts."

I) Mini-quiz

1. Wat is het verschil tussen include en require?
2. Waarom gebruiken we __DIR__?
3. Is dit MVC?

Antwoorden

1. require stopt bij fout
2. Absolute paden
3. Nee

J) Wat je nu kan

- Layout opsplitsen
- Includes correct gebruiken
- Code structureren
- Problemen voorspellen

K) Samenvatting

- Includes verminderen duplicatie
- __DIR__ is verplicht
- Layout is herbruikbaar
- Geen MVC
- Tussenstap geslaagd

MODULE 1 — EINDREFLECTIE

Waarom dit bewust zal falen

Dit systeem breekt zodra:

- je routing nodig hebt
- logica groeit
- data dynamisch wordt
- security belangrijk wordt

⌚ En exact daarom bestaat MVC

MODULE 1 — Wat je nu beheerst

- ✓ Professioneel admin dashboard
- ✓ Tailwind layouts
- ✓ ApexCharts integratie
- ✓ Includes als tussenstap
- ✓ Visuele context voor OOP

MODULE 2 — PHP OOP Fundamentals

We starten **niet abstract**, maar met:

- waarom includes tekortschieten
- hoe classes dit oplossen
- eerste echte OOP-code
- vergelijking met procedurele PHP

Van includes naar classes (zichtbaar en logisch)

Belangrijk We leren OOP **niet theoretisch**. We vertrekken van een probleem dat je **net gevoeld hebt** in MODULE 1.

MODULE 2 — Overzicht

Wat breekt momenteel in onze codebase?

- index.php beslist **alles**
- Layout en “logica” lopen door elkaar
- Geen herbruikbare verantwoordelijkheden
- Geen echte structuur

Wat OOP oplost:

- verantwoordelijkheden scheiden
- gedrag groeperen
- code **begrijpbaar maken voor mensen**
- schaalbaar werken

LES 2.1 — Waarom includes falen (en classes nodig zijn)

A) Doel

- Exact begrijpen **waarom includes niet volstaan**
- Begrijpen **wat een class echt is**
- OOP zien als **oplossing**, niet als truc

B) Nieuwe termen

class

Een **blauwdruk** die gedrag en data samenhoudt.

object

Een **concrete instantie** van een class.

verantwoordelijkheid

Een duidelijk afgebakende taak in de applicatie.

C) Het probleem (concreet, niet theoretisch)

Huidige situatie (MODULE 1)

```
require header  
require sidebar  
require topbar  
require page  
require footer
```

✗ Problemen:

- index.php weet **te veel**
- Geen centrale plek voor:
 - paginakeuze
 - titels
 - data
- Alles is “los”

Wat we eigenlijk willen

In mensentaal:

“Toon het dashboard”

Niet:

“Include deze 5 bestanden in deze volgorde”

D) Vergelijking met procedurele PHP

Procedureel denken

```
$page = 'dashboard';  
if ($page === 'dashboard') {
```

```
    include 'dashboard.php';
}
```

Probleem:

- logica groeit
- if-else spaghetti
- niet uitbreidbaar

OOP-denken (vooruitblik)

```
$controller = new DashboardController();
$controller->render();
```

✓ verantwoordelijkheid duidelijk ✓ gedrag gegroepeerd ✓ leesbaar

E) Belangrijke mentale shift

! Een class is **geen fancy functie**

Een class:

- **houdt state bij**
- **beschermt data**
- **bundelt gedrag**
- **drukt intentie uit**

F) Mini-quiz

1. Wat is het echte probleem met includes?
2. Wat lost een class op?

Antwoorden

1. Geen structuur of verantwoordelijkheid
2. Gedrag + data bundelen

G) Wat je nu kan

- Problemen herkennen
- OOP zien als oplossing
- Niet blind “classes maken”

H) Samenvatting

- Includes lossen layout op
- Ze lossen structuur niet op
- OOP brengt verantwoordelijkheid
- Classes zijn betekenisvol
- We refactoren bewust

LES 2.2 — Eerste class: DashboardPage (zichtbaar resultaat)

A) Doel

- Eerste **echte class**
- Geen abstractie
- Zichtbaar effect in browser

B) Nieuwe termen

property

Een variabele die bij een object hoort.

method

Een functie die bij een class hoort.

C) Concept (zeer concreet)

We maken een class die zegt:

“Ik ben verantwoordelijk voor het dashboard”

Niet voor:

- routing
- auth
- database

Alleen: **dashboard tonen**

D) Click-per-click

Stap 1: Nieuwe map voor classes

/admin
 /classes

Stap 2: Eerste class bestand

admin/classes/DashboardPage.php

PHP verplicht geen hoofdletters voor class-namen.

Deze code werkt technisch ook:

class dashboardpage {}

Maar: **professionele PHP-code doet dit niet.**

De reden is **afspraak en herkenbaarheid**.

```
<?php
declare(strict_types=1);

class DashboardPage
{
    public function render(): void
```

```

    {
        require __DIR__ . '/..../pages/dashboard.php';
    }
}

```

Stap 3: Gebruik class in index.php

```

admin/index.php

<?php
declare(strict_types=1);

error_reporting(E_ALL);
ini_set('display_errors', '1');

require __DIR__ . '/includes/header.php';
require __DIR__ . '/includes/sidebar.php';

require __DIR__ . '/classes/DashboardPage.php';

$page = new DashboardPage();
?>

<main class="flex-1">
    <?php require __DIR__ . '/includes/topbar.php'; ?>
    <?php $page->render(); ?>
</main>

<?php
require __DIR__ . '/includes/footer.php';

```

E) Verwachte output

✓ Exact hetzelfde dashboard ✓ Maar nu via een class

Dat is **bewust**. We veranderen **structuur**, niet uiterlijk.

F) Waarom dit belangrijk is

We hebben nu:

- een **naam** voor gedrag
- een **afgebakende verantwoordelijkheid**
- een plek waar logica kan groeien

G) Veelgemaakte fouten

Fout	Waarom fout
Alles meteen in class steken	Te snel
HTML in class dumpen	Verkeerde verantwoordelijkheid
Denken dat dit al MVC is	Nee

H) Mini-quiz

1. Wat doet render()?
2. Waarom zit HTML niet in de class?

Antwoorden

1. Toont dashboard
2. Scheiding van verantwoordelijkheden

I) Wat je nu kan

- Eerste class schrijven
- Object instantiëren
- Method aanroepen
- Structuur verbeteren

J) Samenvatting (5 bullets)

- Classes introduceren gedrag
- Object roept method aan
- UI blijft ongewijzigd
- Structuur verbetert
- Basis gelegd

LES 2.3 — Properties & constructor (echte OOP)

A) Doel

- Begrijpen wat **state** is
- Data aan object koppelen
- Voorbereiding op dynamische pagina's

B) Nieuwe termen

constructor

Speciale method die wordt uitgevoerd bij new.

state

De interne toestand van een object.

C) Concept

We willen niet hardcoded:

Dashboard

We willen:

```
$pageTitle = 'Dashboard';
```

En dat **gekoppeld** aan het object.

D) Click-per-click

Stap 1: Pas class aan

```
admin/classes/DashboardPage.php

<?php
declare(strict_types=1);

class DashboardPage
{
    private string $title;

    public function __construct(string $title)
    {
        $this->title = $title;
    }

    public function getTitle(): string
    {
        return $this->title;
    }

    public function render(): void
    {
        require __DIR__ . '/../pages/dashboard.php';
    }
}
```

Stap 2: Gebruik constructor

```
admin/index.php

$page = new DashboardPage('Dashboard');
```

Stap 3: Gebruik titel in topbar

```
admin/includes/topbar.php

<header class="bg-white shadow px-6 py-4 flex justify-between">
    <span class="font-semibold">
        <?php echo htmlspecialchars($page->getTitle(), ENT_QUOTES); ?>
    </span>
    <span class="text-sm text-gray-600">Admin</span>
</header>
```

htmlspecialchars is een klassieke en correcte manier om **XSS-aanvallen te voorkomen**.

Wat doet htmlspecialchars()

htmlspecialchars() zet gevaarlijke HTML-tekens om naar veilige tekst.

Het converteert onder andere:

Teken	Wordt
<	<
>	>
"	"
'	'
&	&

Voorbeeld zonder beveiliging

```
$title = '<script>alert("hacked")</script>';
```

echo \$title;

→ Browser voert JavaScript uit → **XSS aanval**

Met htmlspecialchars()

```
echo htmlspecialchars($title, ENT_QUOTES);
```

Output in HTML:

```
&lt;script&gt;alert(&quot;hacked&quot;)&lt;/script&gt;
```

→ Browser toont gewoon tekst

→ Geen JavaScript uitvoering

→ Veilig

Wat betekent ENT_QUOTES?

ENT_QUOTES is een vlag die zegt:

Converteer **zowel dubbele als enkele quotes**

Dus:

Teken	Wordt
"	"
'	'

Zonder ENT_QUOTES worden enkel dubbele quotes omgezet.

Waarom is dit belangrijk?

Zonder ENT_QUOTES kan iemand nog injecteren in HTML-attributen:

```
<input value='<?= $title ?>'>
```

Als \$title bevat:

```
' onmouseover="alert(1)"
```

Dan wordt je HTML:

```
<input value=" onmouseover="alert(1)">
```

→ XSS

Met ENT_QUOTES is dit onmogelijk.

E) Verwachte output

Open je inspect element:

```
<!-- Topbar -->
▼ <header class="bg-white shadow px-6 py-4 flex justify-between">
    <span class="font-semibold"> Dashboard </span> == $0
    <span class="text-sm text-gray-600">Admin</span>
</header>
```

✓ Titel komt uit object ✓ Data zit niet meer "los"

F) Waarom private?

- Beschermt data
- Dwingt gebruik van methods
- Voorkomt rommelcode

G) Veelgemaakte fouten

- public \$title
- constructor overslaan
- echo in class doen

H) Mini-quiz

1. Wat doet een constructor?
2. Waarom is \$title private?

Antwoorden

1. Initialiseert object
2. Encapsulation

I) Wat je nu kan

- State beheren
- Constructor gebruiken
- Data koppelen aan object

J) Samenvatting

- Object heeft state
- Constructor zet state
- Properties zijn private
- Methods geven toegang
- OOP wordt nuttig

Module samenvatting

- Includes waren tussenstap
- Classes introduceren verantwoordelijkheid
- OOP is geen magie

- Structuur groeit logisch
- Klaar voor verdieping