Web Applicaties

Inleiding

Webapplicaties kom je overal tegen: bij het bestellen van eten, het inloggen op school, of het bekijken van je bankrekening. Toch zien de meeste mensen alleen wat er op het scherm gebeurt, terwijl er onder de motorkap veel meer speelt. Een webapplicatie bestaat namelijk uit verschillende onderdelen die samenwerken om gegevens op te slaan, te verwerken en te tonen aan de gebruiker.

In deze les leer je stap voor stap hoe een webapplicatie is opgebouwd. Je ontdekt hoe je echte informatie (zoals een lijst met taken of notities) kunt omzetten naar een digitaal systeem met een database, PHP-code en een nette webpagina met HTML, CSS en JavaScript.

Het gaat in deze lessen niet alleen om het *maken* van code, maar vooral om het *begrijpen* van hoe alles samenwerkt. Je leert waarom een duidelijke opbouw belangrijk is, hoe data wordt opgeslagen, en hoe de logica in de code bepaalt wat er op het scherm verschijnt.

Na deze lessen kun je uitleggen wat een webapplicatie is, hoe de drie lagen (frontend, backend en database) samenwerken, en kun je zelf een eenvoudige maar goed gestructureerde webapplicatie bouwen.

1 Wat is een Webapplicatie?

? Leerdoelen

- Je weet wat een webapplicatie is en waaruit die bestaat.
- Je begrijpt hoe de drie lagen (frontend, backend en database) samenwerken.
- Je kunt uitleggen wat het verschil is tussen data, logica en vormgeving.

? Uitleg

Wat is een webapplicatie?

Een **webapplicatie** is een programma dat in de browser draait en gegevens kan opslaan en verwerken. Denk aan een webshop, een reserveringssysteem of een notitie-app. Zo'n applicatie bestaat meestal uit drie onderdelen (lagen) die samenwerken:

1. **Frontend (voorkant)** – wat de gebruiker ziet en gebruikt, gemaakt met HTML, CSS en JavaScript.
2. **Backend (achterkant)** – de code die bepaalt wat er gebeurt als je op knoppen klikt of formulieren verstuurt. Vaak in PHP of Python.
3. **Database** – hier worden alle gegevens opgeslagen, bijvoorbeeld met MySQL.

Samen zorgen deze lagen ervoor dat een gebruiker iets kan invoeren, dat de code het verwerkt, en dat het resultaat weer op het scherm verschijnt.

Voorbeeld: een restaurant

Je kunt een webapplicatie vergelijken met een restaurant:

- De **frontend** is de ober – die neemt bestellingen op bij de klant.
- De **backend** is de keuken – daar wordt de bestelling klaargemaakt.
- De **database** is de voorraadkast – daar liggen de ingrediënten opgeslagen.

2 Modelleren: denken in onderdelen

? Leerdoelen

- Je kunt uitleggen waarom je eerst nadenkt over onderdelen voordat je gaat programmeren.
- Je kunt belangrijke gegevens (zoals gebruikers en taken) beschrijven in tabellen.
- Je begrijpt hoe tabellen aan elkaar gekoppeld kunnen worden.

? Uitleg

Voordat je begint met programmeren, is het slim om eerst te bedenken **welke informatie** je nodig hebt en hoe die met elkaar samenhangt. Dat heet **modelleren**: je maakt een eenvoudig overzicht van de onderdelen van je applicatie.

Bijvoorbeeld: je wilt een *To-Do app* bouwen waarin een gebruiker taken kan aanmaken. Dan heb je de volgende onderdelen nodig:

- **Gebruiker** – naam, e-mail en wachtwoord.
- **Taak** – titel, beschrijving, status (voltooid of niet) en datum.
- **Categorie** – optioneel, om taken te groeperen (bijv. School, Werk, Persoonlijk).

Eén gebruiker kan meerdere taken hebben. Dat leggen we vast in tabellen in de database.

Van idee naar tabel

Elke onderdeel wordt een **tabel** met kolommen voor de gegevens:

Onderdeel	Kolommen
Gebruikers	id, naam, email, wachtwoord
Taken	id, gebruiker_id, titel, beschrijving, voltooid, datum

SQL: zo maak je tabellen

```
CREATE TABLE gebruikers (
  id INT AUTO_INCREMENT PRIMARY KEY,
  naam VARCHAR(100),
  email VARCHAR(100),
  wachtwoord VARCHAR(255)
);

CREATE TABLE taken (
  id INT AUTO_INCREMENT PRIMARY KEY,
  gebruiker_id INT,
  titel VARCHAR(200),
  beschrijving TEXT,
  voltooid BOOLEAN,
  datum DATE,
  FOREIGN KEY (gebruiker_id) REFERENCES gebruikers(id)
);
```

3 De Backend: de logica

? Leerdoelen

- Je weet wat de backend doet in een webapplicatie.
- Je kunt uitleggen hoe PHP met een database praat.
- Je begrijpt dat de backend de “regels” van de applicatie uitvoert.

? Uitleg

De **backend** is het deel van je webapplicatie dat alles regelt wat de gebruiker niet ziet. Als je op een knop klikt om iets op te slaan, zorgt de backend dat die informatie in de database terechtkomt.

```
<?php
$conn = new mysqli("localhost", "root", "", "todoapp");

$titel = $_POST['titel'];
$beschrijving = $_POST['beschrijving'];

$sql = "INSERT INTO taken (titel, beschrijving) VALUES ('$titel', '$beschrijving')";
$conn->query($sql);
?>
```

Hierboven zie je een simpel PHP-script dat een taak toevoegt aan de database. De backend vertaalt dus wat de gebruiker doet in de browser naar acties in de database.

4 De Frontend: wat de gebruiker ziet

? Leerdoelen

- Je weet wat HTML, CSS en JavaScript doen in een webpagina.
- Je begrijpt het verschil tussen structuur (HTML), vorm (CSS) en gedrag (JS).

? Uitleg

De **frontend** is wat de gebruiker ziet en gebruikt in de browser. Die bestaat uit drie bouwstenen:

HTML (structuur)

```
<form action="add_task.php" method="post">
  <input type="text" name="titel" placeholder="Titel">
```

```
  <textarea name="beschrijving"></textarea>
  <button type="submit">Toevoegen</button>
</form>
```

CSS (stijl)

```
body { font-family: Arial; background: #f8f8f8; }
button { background: #007BFF; color: white; border-radius: 4px; }
```

JavaScript (gedrag)

```
document.querySelector('form').addEventListener('submit', () => {
  alert('Taak toegevoegd!');
});
```

Samen zorgen deze drie talen ervoor dat een webpagina werkt, er goed uitziet en reageert op wat de gebruiker doet.

5 Hoe werken de lagen samen?

? Leerdoelen

- Je begrijpt hoe frontend, backend en database samenwerken.
- Je ziet waarom deze opdeling handig is bij het bouwen van software.

? Uitleg

Alle onderdelen van een webapplicatie werken samen in een vaste volgorde:

1. De gebruiker vult iets in (bijv. een formulier) in de frontend.
2. De backend ontvangt die informatie en slaat het op in de database.
3. De database bewaart de gegevens en stuurt ze terug als de backend ze opvraagt.
4. De backend stuurt de gegevens weer naar de frontend, die ze netjes toont op het scherm.

Omdat elk onderdeel zijn eigen taak heeft, kun je later makkelijk iets aanpassen: de opmaak zonder de logica te veranderen, of de database uitbreiden zonder de pagina’s te herschrijven.

?? Opdracht – Bouw een mini webapplicatie

1. Maak in je database een tabel `notities` met kolommen `id`, `titel` en `tekst`.
2. Schrijf een PHP-bestand `add_note.php` waarin de gebruiker een nieuwe notitie kan opslaan.
3. Maak een tweede pagina `show_notes.php` die alle notities uit de database laat zien.
4. Gebruik CSS om de notities overzichtelijk weer te geven.
5. Laat met JavaScript een melding zien als een notitie is toegevoegd.

? Reflectie

- Waarom is het handig om de code in lagen (frontend, backend, database) op te delen?
- Welke laag vond jij het makkelijkst of leukst om te begrijpen?
- Wat kan er misgaan als je database niet goed is ontworpen?

? Inleveren

- Lever de drie bestanden in: `add_note.php`, `show_notes.php` en de database-export (`notities.sql`).
- Maak een kort reflectieverslag (`reflectie-webapp.txt`) waarin je vertelt wat je geleerd hebt.

6 Het recept voor een webapplicatie ?

Inleiding

Je hebt net een eigen mini-applicatie gemaakt. In deze les ga je leren hoe je dat op een gestructureerde manier kunt aanpakken. We gebruiken hiervoor een duidelijk **stappenplan** – of beter gezegd: een **recept**. Zoals bij koken volg je een vaste volgorde van handelingen om een goed eindresultaat te krijgen.

? Leerdoelen

- Je begrijpt dat het bouwen van een webapplicatie lijkt op het volgen van een recept.

- Je kunt de belangrijkste stappen benoemen: plannen, modelleren, bouwen, testen en verbeteren.
- Je ziet hoe frontend, backend en database samenwerken als ingrediënten van één geheel.

? Uitleg

Een webapplicatie maken lijkt op koken: je hebt ingrediënten, een plan en een goede volgorde nodig. Als je alles tegelijk doet of stappen overslaat, mislukt het gerecht (of de code). Daarom werken we met een **recept** – een vaste volgorde van handelingen.

Het recept □□

1. **Stap 1 – Kies het gerecht (het doel)**
   Bedenk wat je gaat maken: een takenlijst, een webshop of een boekensysteem. Zoals een kok kiest welk gerecht hij kookt, kies jij het doel van je applicatie. □□ Vraag: wat moet de gebruiker ermee kunnen doen?
2. **Stap 2 – Schrijf het boodschappenlijstje (datamodel)**
   Bedenk welke gegevens je nodig hebt. Voor een webshop heb je bijvoorbeeld *producten*, *klanten* en *bestellingen*. Dat noemen we je **datamodel** — een overzicht van alle onderdelen en hun eigenschappen.
3. **Stap 3 – Bereid de ingrediënten voor (database maken)**
   Je zet je datamodel om in tabellen in een database (zoals MySQL). Dat is je voorraadkast met ingrediënten. Je gebruikt SQL om de structuur te maken:

   ```
   CREATE TABLE producten (
     id INT AUTO_INCREMENT PRIMARY KEY,
     naam VARCHAR(100),
     prijs DECIMAL(6,2)
   );
   ```

4. **Stap 4 – Kook de basis (backend logica)**
   De backend (bijv. PHP) is de keuken waar het echte werk gebeurt. Hier wordt data verwerkt, berekeningen gemaakt en met de database gecommuniceerd.

   ```
   <?php
   $conn = new mysqli("localhost", "root", "", "webshop");
   $result = $conn->query("SELECT * FROM producten");
   ?>
   ```

   De backend beslist *hoe* iets moet gebeuren — zoals een kok bepaalt hoe lang iets moet koken.

5. **Stap 5 – Serveer het mooi op (frontend)**
   De frontend is wat de gebruiker ziet — het bord waarop je gerecht wordt geserveerd. Met **HTML** maak je de structuur, met **CSS** de opmaak en met **JavaScript** de interactie.

```
<h1>Producten</h1>
<ul>
  <li>T-shirt – €19.99</li>
  <li>Pet – €14.50</li>
</ul>
```

6. **Stap 6 – Proef en verbeter (testen)**
   Bekijk je applicatie in de browser. Werkt alles zoals bedoeld? Kun je gegevens opslaan, tonen, en aanpassen? Net als bij koken proef je tussendoor en pas je aan wat nog niet goed smaakt.
7. **Stap 7 – Serveer aan de gasten (opleveren)**
   Als alles goed werkt, kun je je webapplicatie uploaden naar een echte server. Daarmee kunnen anderen (je gebruikers) het “proeven” — oftewel: gebruiken.

?? Opdracht – Jouw eigen recept

1. Kies een onderwerp (bijv. receptenboek, studentenlijst, boekenoverzicht).
2. Maak een kort **datamodel** met 2 of 3 tabellen (bijv. *studenten*, *opleidingen*).
3. Bedenk in één alinea wat de applicatie moet kunnen doen.
4. Schrijf jouw “recept” in stappen 1 t/m 7 — kort en in je eigen woorden.
5. Voeg eventueel kleine stukjes voorbeeldcode of SQL toe om het concreter te maken.

? Reflectie

- Welke stap vond jij het moeilijkst: plannen, modelleren of programmeren?
- Waarom is het belangrijk om eerst het recept te bedenken vóór je begint met coderen?
- Hoe helpt dit stappenplan je om fouten sneller te vinden of te verbeteren?

? Inleveren

- Lever in één document in:

    - Een korte beschrijving van het doel van je applicatie
    - Je datamodel (schema of tabel)
    - Je eigen “recept” in stappen 1 t/m 7
- Bestandsnaam: `recept-webapplicatie-<jouwnaam>.pdf`

7? Zo doen professionals het

(Voor studenten die meer uitdaging zoeken en die zich willen voorbereiden op het gebruiken van Frameworks)

In deze les heb je geleerd hoe je handmatig een webapplicatie kunt opbouwen: database, backend en frontend. In de praktijk gebruiken professionele ontwikkelaars extra technieken en hulpmiddelen om hun werk sneller, veiliger en beter onderhoudbaar te maken. Hieronder lees je enkele verdiepende onderwerpen waarmee je je kennis kunt uitbreiden.

1. Frameworks

In plaats van alles vanaf nul te bouwen, gebruiken ontwikkelaars vaak een **framework**. Een framework is een verzameling kant-en-klare onderdelen die helpen bij het structureren van code. Voorbeelden:

- **Laravel** (PHP) – biedt tools voor routing, beveiliging en database-beheer.
- **Django** (Python) – automatiseert veel terugkerende taken.
- **React** of **Vue.js** (JavaScript) – voor dynamische frontends.

Het voordeel van frameworks is dat ze een duidelijke structuur afdwingen, waardoor meerdere ontwikkelaars tegelijk aan één project kunnen werken.

2. MVC-structuur

Bij grotere applicaties wordt de code vaak opgedeeld volgens het **MVC-model**:

- **Model** – de data en database-logica.
- **View** – wat de gebruiker ziet (de HTML-pagina’s).
- **Controller** – de logica die bepaalt wat er gebeurt bij elke actie.

Door deze scheiding blijft de code overzichtelijk en kun je makkelijker fouten opsporen of functies uitbreiden. Veel frameworks, zoals Laravel of Django, werken standaard op deze manier.

3. API’s en JSON

Soms communiceren webapplicaties niet alleen met een eigen database, maar ook met andere systemen. Dat gebeurt via een **API** (Application Programming Interface). De gegevens worden meestal uitgewisseld in het **JSON-formaat**:

```
{
  "titel": "Nieuwe taak",
  "beschrijving": "Maak de HTML les af",
  "voltooid": false
}
```

Met een API kun je bijvoorbeeld data ophalen van een weer-, nieuws- of schoolserver, en die gebruiken in jouw eigen applicatie.

4. Beveiliging en validatie

Goede webapplicaties zijn niet alleen mooi en handig, maar ook **veilig**. Dat betekent onder andere:

- Controleer altijd invoer van gebruikers (bijv. met `htmlspecialchars()` in PHP).
- Gebruik wachtwoorden nooit als platte tekst – versleutel ze met `password_hash()`.
- Gebruik `prepare statements` om SQL-injectie te voorkomen.

5. Versiebeheer en samenwerking

Professionals gebruiken systemen zoals **Git** (met GitHub of GitLab) om hun code te bewaren en te delen. Hiermee kun je zien wie wat heeft aangepast en kun je veilig samenwerken in één project.

□□ Tip voor gevorderden

Wil je deze onderwerpen echt oefenen? Probeer jouw mini-applicatie om te bouwen tot een **klein MVC-project** of maak een eenvoudige **REST-API** die JSON-data verstuurt en ontvangt. Zo ervaar je hoe professionele webontwikkelaars te werk gaan.

8 De ontwikkelomgeving ?

? Leerdoelen

- Je weet welke tools je nodig hebt om een webapplicatie te ontwikkelen.

- Je begrijpt de rol van Visual Studio Code, XAMPP en de browser.
- Je kunt uitleggen hoe deze drie samen werken tijdens het testen van je code.

? Uitleg

Om een webapplicatie te bouwen heb je niet alleen code nodig, maar ook een **ontwikkelomgeving** — de verzameling van programma's waarin je code schrijft, uitvoert en bekijkt. Die omgeving bestaat meestal uit drie onderdelen:

1. **Visual Studio Code (VSC)** – de werktafel
2. **XAMPP** – de keuken (de server en database)
3. **De browser** – het bord waarop je het resultaat ziet

1. Visual Studio Code □□□ – Hier schrijf je de code

Visual Studio Code (afgekort: VSC) is een **code-editor**. Het is te vergelijken met Word, maar dan speciaal gemaakt voor programmeurs. Je schrijft hier je *HTML*, *CSS*, *JavaScript* en *PHP*-bestanden. VSC helpt je met:

- Kleuren en opmaak van code (syntax highlighting)
- Fouten sneller vinden (linting en meldingen)
- Plugins voor PHP, MySQL of HTML live server

□□ *Je gebruikt VSC om te **maken** en te **bewaren** wat jouw webapplicatie doet.*

> Er bestaan tegenwoordig clonen van VSC waarin AI tools ingebakken zitten, bijvoorbeeld **Cursor** of **Windsurf**

2. XAMPP □□ – Hier draait de webserver

Een webapplicatie heeft een **webserver** nodig — een programma dat jouw PHP-bestanden uitvoert en communiceert met de database. Normaal gesproken staat die server ergens op het internet, maar tijdens het ontwikkelen gebruik je een lokale versie op je eigen laptop: **XAMPP**.

XAMPP bevat drie belangrijke onderdelen:

- **Apache** – de webserver die jouw PHP-code uitvoert;
- **MySQL / MariaDB** – de database waarin je gegevens opslaat;

- **phpMyAdmin** – een handige webpagina om de database te beheren.

Wanneer je in XAMPP op “Start” drukt bij Apache en MySQL, start je eigenlijk jouw eigen kleine internetserver op je computer. Daarom open je je projecten via `http://localhost` in plaats van via een gewoon bestandspad.

□□ *Je gebruikt XAMPP om je applicatie **uit te voeren** en de database lokaal te laten werken.*

3. De browser □□ – Hier test je de webapp

De browser (zoals Chrome, Edge of Firefox) is waar je jouw webapplicatie **ziet** en **test**. De browser haalt de bestanden op bij XAMPP (de server), voert de JavaScript uit en toont het resultaat van jouw PHP-code en databasegegevens.

In de browser kun je met de **ontwikkelaarstools (F12)** zien wat er “onder de motorkap” gebeurt:

- HTML-structuur bekijken
- CSS live aanpassen
- Fouten in JavaScript zien (Console)
- Verkeer tussen frontend en backend volgen (Netwerk-tab)

□□ *Je gebruikt de browser om je werk te **testen** en **controleren**.*

□□ Hoe werkt het samen?

De drie tools werken als één geheel:

1. Je schrijft de code in **Visual Studio Code**.
2. XAMPP voert de PHP-bestanden uit en praat met de **database**.
3. De **browser** laat het resultaat zien via `http://localhost`.

Dat proces herhaalt zich steeds: je past iets aan in VSC, vernieuwt de pagina in de browser, en XAMPP zorgt dat de nieuwe code wordt uitgevoerd. Zo ontwikkel je stap voor stap een webapplicatie.

?? Opdracht – Alles laten samenwerken

1. Installeer Visual Studio Code en XAMPP (als dat nog niet is gebeurd).
2. Maak in je XAMPP-map (`htdocs`) een nieuwe map `mijnapp`.

Revision #7
Created 2025-10-19 14:54:41 UTC by Max
Updated 2025-10-19 15:25:04 UTC by Max