# Projectdocumentatie - Group_Project_Webtechnologie

## 📘 Overzicht

Dit project is ontwikkeld in het kader van het vak Webtechnologie en omvat een volledige webapplicatie die bestaat uit een frontend, backend, RESTful API, en een PostgreSQL-database. De applicatie demonstreert een eenvoudige gebruikersauthenticatie met veilige opslag van gebruikersgegevens en communicatie tussen client en server via API-endpoints.

## 📁 Structuur van de repository

De projectstructuur is als volgt opgebouwd:

| Bestand/Map | Beschrijving |
|---|---|
| **/api/** | Bevat de backend API-bestanden zoals login.php, register.php, enz. |
| **app.js** | Frontend JavaScript die AJAX-verzoeken naar de API verstuurt. |
| **index.php** | De startpagina van de webapplicatie met dynamische content via PHP. |
| **caddyfile** | Caddy server configuratiebestand voor lokale HTTPS routing. |
| **.gitignore** | Bepaalt welke bestanden niet worden opgenomen in versiebeheer. |
| **HTTPS/** | Eventuele certificaatbestanden en HTTPS-gerelateerde configuratie. |
| **README.md** | Toelichting van het project, installatie-instructies en to-do lijst. |
| **package.json** | Node.js projectmetadata voor de frontend of API logic. |
| **package-lock.json** | Bevat exacte versies van geïnstalleerde npm dependencies. |

## ⚙️ Installatie en Configuratie

1. Installeer benodigde pakketten:
sudo apt update
sudo apt install php8.3-fpm caddy postgresql postgresql-contrib php-pgsql net-tools apache2 php npm

2. Start de noodzakelijke services:
sudo systemctl start php8.3-fpm
sudo systemctl start postgresql

3. Initieer de Node.js omgeving (voor API logic):
cd api
npm init -y

4. Start de lokale API server:
php -S localhost:8000

5. Start de webserver (Caddy):
sudo caddy run --config /pad/naar/caddyfile --adapter caddyfile

## 🗃️ Database Setup

1. PostgreSQL installatie en configuratie:
sudo apt install postgresql postgresql-contrib php-pgsql

2. Maak een nieuwe gebruiker en database aan:
sudo -u postgres psql
CREATE USER "WebTechUser" WITH PASSWORD 'Abracadabra is a magic word! ;)';
CREATE DATABASE webtechname;
GRANT ALL PRIVILEGES ON DATABASE webtechname TO "WebTechUser";
ALTER DATABASE webtechname OWNER TO "WebTechUser";
GRANT USAGE, CREATE ON SCHEMA public TO "WebTechUser";
ALTER SCHEMA public OWNER TO "WebTechUser";

3. Maak de gebruikers-tabel aan:
```
CREATE TABLE users (
   id SERIAL PRIMARY KEY,
   username TEXT NOT NULL UNIQUE,
   password TEXT NOT NULL
);
```

4. Voeg een gebruiker toe voor testdoeleinden:
INSERT INTO users (username, password) VALUES ('admin', 'wachtwoord');

## 🗒️ Uitleg van Commits

Elke commit in de repository is bedoeld om een specifieke toevoeging of verbetering aan te brengen. Hieronder volgt een overzicht van belangrijke commits:

- "caddyfile root verandert voor mijn pc": Aanpassing aan de Caddyfile om deze compatibel te maken met de lokale omgeving van een teamlid.
- "added gitignore and ignored some files": Toevoegen van .gitignore om tijdelijke bestanden en build-artifacts buiten Git te houden.
- "Werkend met SQL en DATABASE": Eerste functionele implementatie van database-integratie via PHP.
- "Update README.md": Bijwerken van de documentatie en voortgang binnen het project.

Info

Wat is api?

**API (Application Programming Interface)** is een manier waarop programma's met elkaar communiceren. In webtechnologie wordt het vaak gebruikt om gegevens op te vragen van een andere dienst via het internet, bijvoorbeeld het weer of gebruikersdata.

Een web-API gebruikt meestal:

- HTTP-verzoeken (zoals GET of POST)
- JSON-formaat voor data
- Endpoints (specifieke URL's)
- Soms een API-sleutel voor toegang

Wat is .GitIgnore

**.gitignore** is een bestand dat Git vertelt welke bestanden of mappen **niet** in de repository moeten komen, zoals tijdelijke bestanden, build-mappen (zoals dist/), of dependencies (node_modules/). Zo blijft je project netjes en zet je geen onnodige bestanden online.

Wat doet een licensie?

Een licentie bepaalt wat anderen wel of niet mogen doen met jouw code of website. Het geeft aan of iemand de code mag kopiëren, aanpassen of gebruiken, en onder welke voorwaarden. Zonder licentie mag niemand de code zomaar gebruiken. Met een open source licentie, zoals de MIT- of GPL-licentie, mag dat wel, maar vaak met regels zoals naamsvermelding of het verplicht delen van aangepaste versies. Zo zorgt een licentie ervoor dat de rechten en plichten rond jouw project duidelijk zijn.

Wat staat er allemaal in de .ReadMe file?

Een **README-bestand** is vaak het eerste bestand dat iemand ziet bij een project. Hierin leg je uit **wat het project is, hoe het werkt en hoe je het kunt gebruiken**.

Meestal bevat een README:

- Een korte **beschrijving** van het project
- **Installatie-instructies**
- Uitleg over **gebruik**
- Eventueel: licentie, auteurs en contactinformatie

Zo helpt de README anderen (en jezelf) om snel te snappen wat het project doet en hoe ze ermee kunnen werken.

Wat is de Caddy File?

Een **Caddyfile** is een configuratiebestand voor de **Caddy webserver**. Hierin staat hoe de server jouw website of webapp moet draaien.

In een Caddyfile bepaal je bijvoorbeeld:

- Op welk **domein of poort** de site draait
- Of de site automatisch **HTTPS** krijgt
- Waar de **bestanden** van de website staan
- Eventueel: **omleidingen (redirects)**, omgevingsvariabelen, reverse proxies, etc.

**Waarom is dit belangrijk?**

De Caddyfile zorgt ervoor dat je website veilig, correct en automatisch online komt. Caddy regelt hierdoor zelf HTTPS-certificaten, wat je bij andere servers vaak handmatig moet doen.

Wat is Index.hpp?

Het bestand **index.php** is vaak de **startpagina** van een PHP-website. Als iemand jouw site bezoekt, zoekt de server automatisch naar index.php om te tonen.

**Wat doet het?**

- Het **start** je website of webapplicatie
- Voert **PHP-code** uit om pagina's dynamisch te maken (bijv. met gebruikers, formulieren of databases)
- **Laadt andere onderdelen**, zoals scripts, templates of routes

**Waarom is dit belangrijk?**

Zonder index.php weet de server niet wat hij moet tonen. Het is dus de **ingang** van je site.

**Wat staat er meestal in?**

- HTML + PHP
- Logica voor routing of opbouw van de pagina
- Koppeling met een database
- Laad instructies voor CSS of JavaScript

Wat is Login.php?

Het bestand **login.php** wordt gebruikt om **gebruikers te laten inloggen** op een website.

**Wat doet het?**

- Controleert de **ingevoerde gebruikersnaam en wachtwoord**
- Vergelijkt deze gegevens met de data uit een **database**
- Start een **sessie** als de gegevens kloppen
- Geeft een foutmelding als het fout is

**Waarom is dit belangrijk?**

Zonder login.php kan je website geen beveiligde login aanbieden. Dit bestand is dus nodig om **toegang tot afgeschermde delen** van de site te regelen, zoals een dashboard.

**Wat staat er meestal in?**

- PHP-code om gegevens uit een **formulier (POST)** op te halen
- Connectie met een **database**
- **Validatie** van de invoer
- Start van een **sessie** (bij succes) of een foutmelding (bij mislukking)

Wat is package-lock.json?

Het bestand **package-lock.json** wordt automatisch aangemaakt door **npm** (Node Package Manager) als je afhankelijkheden (packages) installeert.

**Wat doet het?**

- Legt precies vast **welke versies** van alle packages (en hun sub-packages) zijn geïnstalleerd.
- Zorgt dat anderen exact dezelfde versie van de code krijgen als ze het project installeren.

**Waarom is dit belangrijk?**

Zonder package-lock.json kan het gebeuren dat iemand anders nieuwere (en mogelijk incompatibele) versies van packages binnenhaalt. Dit bestand zorgt dus voor **betrouwbare en stabiele builds**.

**Wat staat erin?**

- Namen en **versies van alle packages**
- Download-URL's
- Hashes (voor controle op fouten of manipulatie)

Je hoeft dit bestand meestal niet handmatig aan te passen — npm beheert het automatisch.

Wat is package.json?

Het bestand **package.json** is het **hart van een Node.js-project**. Hierin staat alle belangrijke informatie over je project en zijn afhankelijkheden.

**Wat doet het?**

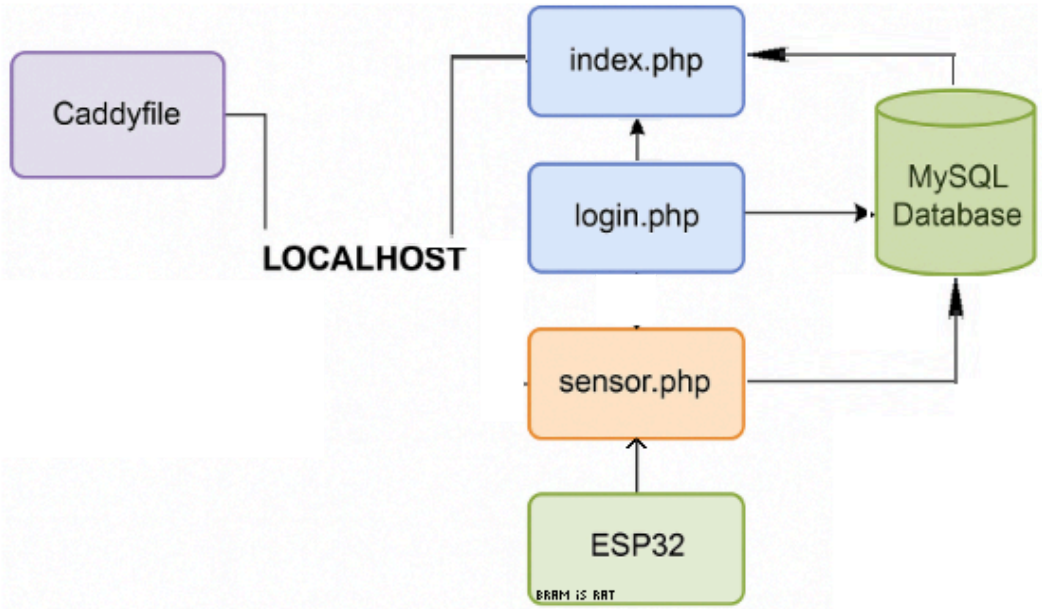
- Beschrijft het project: **naam, versie, beschrijving, auteur**
- Lijst de **nodige packages** op (dependencies)
- Bevat eventuele **scripts** om taken automatisch uit te voeren (zoals starten of bouwen van de app)

**Waarom is dit belangrijk?**

Zonder package.json weet npm niet welke dependencies of scripts het project nodig heeft. Het maakt je project **herbruikbaar en deelbaar** met anderen.

**Wat staat erin?**

- Projectinformatie (naam, versie)
- **Dependencies** (zoals React, Express, etc.)
- Scripts (zoals npm start, npm run build)




## 🔧 1. Caddyfile

- De **Caddyfile** fungeert als de configuratie voor de webserver (**Caddy**) op jullie localhost.
- Hij bepaalt hoe verzoeken naar de juiste bestanden (zoals `index.php`, `login.php`, ...) worden geleid.
- Alles wat je in je browser opent via `localhost`, wordt door de Caddyfile verwerkt.

---

## 📄 2. index.php

- Dit is de hoofdpagina van de site.
- Aanvankelijk toonde dit bestand een eenvoudige "Hello World", later bevatte het ook:
    - Een **aftellende timer** tot Bram zijn verjaardag.
    - HTML-elementen voor layout.

- De pagina is alleen toegankelijk nadat je succesvol bent ingelogd via de API.

---

## 🔐 3. login.php (RESTful API)

- Hier hebben jullie een **RESTful API** gemaakt in PHP.
- Het controleert gebruikersinvoer (gebruikersnaam & wachtwoord) en vergelijkt deze met de gegevens uit de database.
- Als de login klopt, stuurt deze een bevestiging terug die toegang verleent tot `index.php`.

---

## 💾 4. MySQL Database

- De database bevat een tabel met gebruikersnamen en wachtwoorden.
- `login.php` maakt verbinding met deze database om logingegevens te verifiëren.
- Later kan hier ook data van de sensor in opgeslagen worden, indien gewenst.

---

## 🔌 5. sensor.php & ESP32

- Jullie hebben een **ESP32** gebruikt in combinatie met een **temperatuur- en luchtvochtigheidssensor**.
- Deze maakt via `sensor.php` verbinding met de localhost op poort 8001.
- De gemeten data kan eventueel doorgegeven worden aan de database of zichtbaar gemaakt worden op de website.

---

## 🖥️ LOCALHOST als kern

- Alle componenten draaien op jullie lokale server (localhost).
- Zowel de webpagina als de communicatie met de sensor en database gebeurt binnen deze lokale omgeving.

---

# De Vragen en Antwoorden van alle AI’s, deze vind ik beetje gek soms de vragen.

## Web Server Setup & Configuration

1. Wat is het doel van een webserver zoals Caddy in een webapplicatie?

Antwoord:

Een webserver zoals Caddy host de website, serveert statische bestanden en verwerkt dynamische inhoud via server-side scripts zoals PHP. Het luistert op poorten 80 (HTTP) en 443 (HTTPS) om inkomende verbindingen te accepteren.

2. Hoe configureer je een webserver om te luisteren op poorten 80 en 443?

Antwoord:

In de Caddyfile zet je de domeinen of localhost en specificeer je dat de server moet luisteren op poorten 80 en 443. Caddy doet dit standaard voor HTTP en HTTPS.

3. Welke webserver gebruiken jullie in dit project en waarom?

Antwoord:

We gebruiken Caddy vanwege zijn automatische HTTPS-configuratie, eenvoud in setup, en ingebouwde ondersteuning voor certificaatbeheer via Let's Encrypt.

4. Hoe configureer je Caddy om automatische HTTPS te gebruiken?

Antwoord:

Door een domeinnaam op te geven in de Caddyfile, gebruikt Caddy automatisch ACME (Let's Encrypt) om certificaten aan te vragen, te installeren en te vernieuwen.

5. Jullie Caddyfile luistert naar localhost en 127.0.0.1. Waarom zijn beide nodig?

Antwoord:

localhost is de hostnaam die naar 127.0.0.1 wijst. Beide zorgen ervoor dat de server op zowel de naam als het IP-adres reageert, wat best practice is voor compatibiliteit.

6. Wat doet de file_server directive in de Caddyfile?

Antwoord:

Deze instructie vertelt Caddy om statische bestanden te serveren uit de geconfigureerde root directory.

7. Wat betekent de root * regel in de Caddyfile?

Antwoord:

Het stelt de document root in, oftewel de map waar Caddy de bestanden vandaan haalt om te serveren.

8. Wat gebeurt er als je meerdere root directives hebt in dezelfde serverblock?

Antwoord:

Caddy verwerkt directives sequentieel. Meerdere root directives kunnen leiden tot verwarring of fouten; meestal wordt de laatste gebruikt of veroorzaakt een configuratiefout.

9. Wat zijn de verschillen tussen PHP via TCP-socket en via Unix socket?

Antwoord:

Een TCP-socket gebruikt netwerkpoorten en is minder snel voor lokale verbindingen; een Unix socket is sneller en veiliger omdat het geen netwerkverkeer gebruikt en alleen lokaal toegankelijk is.

10. Hoe controleer je of Caddy en PHP-FPM correct draaien via SSH?

Antwoord:

Gebruik:

sudo systemctl status caddy

sudo systemctl status php8.3-fpm

en bekijk de logs met journalctl -u caddy en journalctl -u php8.3-fpm.

# PHP & Backend Logic

11. Wat is het doel van session_start() in PHP?

Antwoord:

Het initializeert of hervat een sessie, zodat gebruikersgegevens (zoals loginstatus) kunnen worden opgeslagen en gedeeld tussen pagina's.

12. Waarom gebruiken jullie PDO in plaats van mysqli of mysql_ functies?

Antwoord:

PDO biedt een uniforme interface voor meerdere databases, ondersteunt prepared statements voor beveiliging, en heeft betere foutafhandeling.

13. Hoe voorkom je SQL-injecties in je PHP-code?

Antwoord:

Door gebruik te maken van prepared statements met bound parameters, zoals $pdo->prepare().

14. Waarom is het belangrijk om wachtwoorden te hashen in de database?

Antwoord:

Omdat platte tekst wachtwoorden kwetsbaar zijn bij datalekken. Wachtwoorden moeten worden gehasht met sterke algoritmes zoals password_hash().

15. Hoe maak je een verbinding met een PostgreSQL database in PHP?

Antwoord:

Door een PDO-object te maken met de DSN pgsql:host=...;port=...;dbname=..., samen met gebruikersnaam en wachtwoord.

16. Wat doet PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION?

Antwoord:

Het zorgt ervoor dat PDO fouten als exceptions werpen, zodat je deze kunt opvangen met try-catch blokken.

17. Hoe wordt sessiebeheer in PHP toegepast in jullie project?

Antwoord:

Door session_start() te gebruiken en sessievariabelen zoals $_SESSION['username'] te gebruiken om de loginstatus te bewaren.

18. Hoe wordt de laatste sensorwaarde uit de database gehaald?

Antwoord:

Door een query met ORDER BY timestamp DESC LIMIT 1, zodat de meest recente wordt opgehaald.

19. Waarom wordt htmlspecialchars() gebruikt bij het tonen van gebruikersgegevens?

Antwoord:

Om Cross-Site Scripting (XSS) te voorkomen door speciale tekens te escapen.

20. Hoe gaat de login-API om met incorrecte inloggegevens?

Antwoord:

Door een 401 status te sturen en een JSON-bericht met foutmelding.

# Frontend & JavaScript

21. Hoe wordt de countdown timer in JavaScript gemaakt?

Antwoord:

Met setInterval() dat elke seconde de resterende tijd tot de verjaardag berekent en weergeeft.

22. Hoe wordt de countup timer in JavaScript gemaakt?

Antwoord:

Door de tijd sinds een startdatum te berekenen en elke seconde bij te werken met setInterval().

23. Waarom wordt setInterval() gebruikt in de timers?

Antwoord:

Omdat het periodiek functies aanroept, waardoor de timers elke seconde worden bijgewerkt.

24. Wat doet htmlspecialchars() in JavaScript-frontend code?

Antwoord:

Het voorkomt XSS door gebruikersinput veilig weer te geven in HTML, meestal wordt dit in PHP gedaan, niet in JavaScript.

25. Wat is het voordeel van CSS-inlining versus externe CSS?

Antwoord:

Inline CSS (in de <style> tag) vereist geen extra HTTP-verzoek, maar is minder overzichtelijk en minder herbruikbaar dan externe CSS-bestanden.

Database & Data Management

26. Welke database gebruiken jullie en waarom?

Antwoord:

PostgreSQL, vanwege de geavanceerde functies, betrouwbaarheid en ondersteuning voor complexe queries.

27. Wat is CRUD en hoe wordt het toegepast in jullie project?

Antwoord:

Create, Read, Update, Delete. Bijvoorbeeld: sensor data toevoegen (Create), ophalen (Read). Update en Delete worden niet momenteel gebruikt maar zijn onderdeel van het principe.

28. Waarom is LIMIT 1 belangrijk bij het ophalen van sensor data?

Antwoord:

Het zorgt dat alleen de meest recente meting wordt opgehaald, wat efficiënt is en relevant voor de gebruiker.

29. Hoe zou je paginering toevoegen voor sensor data?

Antwoord:

Met SQL LIMIT en OFFSET, bijvoorbeeld LIMIT 10 OFFSET 20, gecombineerd met navigatieknoppen in de frontend.

30. Wat is het risico van wachtwoorden in platte tekst?

Antwoord:

Bij datalekken zijn alle wachtwoorden direct leesbaar. Oplossing: hash ze met password_hash().

# API & RESTful Principles

31. Wat betekent 'RESTful' in API-design?

Antwoord:

Het is een architectuurstijl die gebruikmaakt van HTTP-methoden, resource-georiënteerde URL's, stateless communicatie en standaard dataformaten zoals JSON.

32. Welke HTTP-methoden worden gebruikt in jullie API?

Antwoord:

GET voor ophalen, POST voor aanmaken/authenticatie, PUT voor bijwerken, DELETE voor verwijderen.

33. Waarom stuurt login.php JSON terug?

Antwoord:

Omdat het een API-endpoint is, bedoeld voor programma's of frontends die JSON kunnen parsen, niet voor direct tonen in de browser.

34. Hoe beveiligen jullie API tegen brute force-aanvallen?

Antwoord:

Door rate limiting, CAPTCHA, of API-keys/JWT tokens.

35. Wat is het verschil tussen een webpagina en een API-endpoint?

Antwoord:

Een webpagina wordt weergegeven voor de gebruiker; een API-endpoint biedt data voor andere systemen of frontend-applicaties.

# Security & Best Practices

36. Hoe voorkom je SQL-injecties?

Antwoord:

Door gebruik te maken van prepared statements met PDO en input te ontsmetten.

37. Hoe voorkom je XSS-aanvallen?

Antwoord:

Door user input te ontsmetten met htmlspecialchars() en output te encoderen.

38. Hoe beveilig je je API tegen CSRF?

Antwoord:

Met CSRF-tokens die in formulieren worden gebruikt en worden gecontroleerd bij elke POST.

39. Waarom is HTTPS belangrijk?

Antwoord:

Omdat het de communicatie versleutelt, waardoor gegevens niet door derden kunnen worden gelezen of gewijzigd.

40. Hoe stel je de juiste permissies in op de sessiemap?

Antwoord:

Door de map waar PHP sessies worden opgeslagen (bijv. /var/lib/php/sessions of /tmp) de juiste rechten te geven, zodat PHP ze kan lezen en schrijven.

# Version Control & Collaboration

41. Hoe hebben jullie Git gebruikt in het project?

Antwoord:

Door regelmatig commits te maken, branches te gebruiken voor features, en samen te werken via pull requests en issues.

42. Waarom is regelmatige commit-historie belangrijk?

Antwoord:

Omdat het de voortgang documenteert, makkelijk terug te keren is en conflicten beter beheersbaar blijven.

43. Hoe nodig je een collega uit als collaborator op GitHub?

Antwoord:

Door via de repository-instellingen hen uit te nodigen onder "Manage Access" met hun GitHub username.

44. Hoe los je merge conflicts op?

Antwoord:

Door conflicterende bestanden handmatig te bewerken, te committen en de merge compleet te maken.

45. Wat is het voordeel van feature branches?

Antwoord:

Ze isoleren nieuwe functionaliteit, zorgen voor stabiele main branches en vergemakkelijken samenwerking.

# External Device & IoT Integratie

46. Hoe zou een extern embedded device data uploaden?

Antwoord:

Door HTTP POST-verzoeken te sturen met JSON-data naar een API-endpoint zoals /api/sensor-data.

47. Waarom moet het device op een andere machine staan dan de webserver?

Antwoord:

Dit simuleert een realistische IoT-omgeving en test de netwerkcommunicatie over het internet.

48. Welke data zou het device periodiek uploaden?

Antwoord:

Temperatuur, vochtigheid, en timestamp, zoals getoond in de sensor_data tabel.

49. Hoe beveilig je API-toegang voor externe devices?

Antwoord:

Door API-keys, JWT-tokens, of andere authenticatiemethoden te gebruiken.

50. Hoe voorkom je dat te vaak data wordt geüpload?

Antwoord:

Door rate limiting of authenticatie met device-ID en tijdstempels.

# Algemene Web & Project Management

51. Wat zijn de belangrijkste technologieën die jullie gebruiken?

Antwoord:

Caddy (webserver), PHP (server-side scripting), PostgreSQL (database), HTML/CSS/JavaScript (frontend).

52. Waarom moet je user input correct afhandelen?

Antwoord:

Om beveiligingsrisico's zoals SQL-injecties en XSS te voorkomen, en om data-integriteit te waarborgen.

53. Hoe documenteer je je project?

Antwoord:

Door gebruik te maken van README-bestanden, commentaar in de code, en technische documentatie over API's en architectuur.

54. Hoe zou je je project uitbreiden?

Antwoord:

Door functies zoals wachtwoord hashing, admin dashboards, grafieken voor sensor data, of real-time updates toe te voegen.

55. Wat is het belang van goede Git-praktijken?

Antwoord:

Ze zorgen voor overzicht, samenwerking, en het veiligstellen van codeveranderingen.

# En dit zijn de 50 vragen van chatgpt:

---

## 🔧 Webserver & Deployment

1. **Wat doet het `caddyfile` in jullie project?**
   *Het configureert de Caddy-webserver om PHP-bestanden via PHP-FPM te serveren en stelt het root-pad van het project in.*

2. **Op welke poorten moet een webserver luisteren?**
   *Poort 80 voor HTTP en poort 443 voor HTTPS.*

3. **Wat is het voordeel van Caddy boven Apache of Nginx?**
   *Caddy configureert automatisch HTTPS, is makkelijker in gebruik en heeft ingebouwde ondersteuning voor PHP.*

4. **Hoe stel je een Caddy-server in om met PHP samen te werken?**
   *Gebruik `php_fastcgi` en wijs het pad naar de juiste socket of poort van PHP-FPM.*

5. **Waarom gebruiken jullie `php_fastcgi` i.p.v. een gewone socket verbinding?**
   *Omdat het zorgt voor efficiënte communicatie tussen de Caddy-server en PHP-FPM.*

---

## 🧠 PHP & Server-side Logic

6. **Wat doet `session_start()` in PHP?**
   *Het start een nieuwe sessie of hervat een bestaande sessie, waarmee gebruikersinformatie bewaard wordt.*

7. **Hoe beschermen jullie tegen SQL-injecties?**
   *Door gebruik te maken van prepared statements via PDO.*

8. **Wat gebeurt er als de databaseconnectie faalt?**
   *Er wordt een foutmelding weergegeven, maar zonder technische details (voor veiligheid).*

9. **Hoe wordt data uit de database opgehaald in jullie `index.php`?**
   *Met een SQL-query die de laatste temperatuur- en vochtigheidswaarden ophaalt.*

10. **Waarom controleren jullie of `$_SERVER['REQUEST_METHOD'] === 'POST'`?**
    *Om zeker te zijn dat de login alleen via een POST-aanvraag gebeurt en niet via GET.*

---

## 🌐 RESTful API

11. **Wat is een RESTful API?**
    *Een API die gebruik maakt van HTTP-methoden (GET, POST, PUT, DELETE) en gestandaardiseerde URL's.*

12. **Welke HTTP-methoden gebruiken jullie?**
    *Voor login wordt POST gebruikt; voor sensor data waarschijnlijk ook POST.*

13. **Wat gebeurt er als iemand een GET-verzoek stuurt naar `login.php`?**
    *Er wordt een 405 "Method Not Allowed" fout teruggestuurd.*

14. **Wat is het verschil tussen `login.php` en `index.php`?**
    *`login.php` is een API-endpoint voor JSON, `index.php` is de frontend pagina.*

15. **Wat is het voordeel van JSON in een REST API?**
    *Het is lichtgewicht, gemakkelijk te parsen, en breed ondersteund door frontend-technologieën.*

---

## 💻 Frontend (HTML, CSS, JS)

16. **Welke frontend technologieën gebruiken jullie?**
    *HTML, CSS en JavaScript.*

17. **Wat doet de countdown timer op jullie site?**
    *Hij telt af naar een toekomstige datum, bijvoorbeeld een verjaardag.*

18. **Wat doet de countup timer?**
    *Hij telt het aantal dagen sinds een specifieke gebeurtenis (zoals een startdatum).*

19. **Hoe is jullie loginformulier beveiligd?**
    *Via POST-verzoek, server-side validatie, en sessiebeheer.*

20. **Waarom gebruiken jullie `htmlspecialchars()` bij output?**
    *Om XSS (Cross-Site Scripting) te voorkomen.*

---

## 🗃️ Database & Security

21. **Welke database gebruiken jullie?**
    *PostgreSQL.*

22. **Hoe maak je verbinding met de database via PDO?**
    *Via een DSN-string en inloggegevens met foutafhandeling met try-catch.*

23. **Wat is een prepared statement en waarom gebruiken jullie het?**
    *Een voorbereide SQL-query die SQL-injecties voorkomt.*

24. **Waar slaan jullie de sensorgegevens op?**
    *In een tabel genaamd `sensor_data`.*

25. **Wat gebeurt er als de `sensor_data`-tabel leeg is?**
    *Er wordt een melding getoond dat er geen data beschikbaar is.*

---

## 🔐 Beveiliging & Inputvalidatie

26. **Wat zijn voorbeelden van beveiligingsmaatregelen in jullie code?**
    *Session management, input sanitization, prepared statements, error-handing.*

27. **Waarom is het belangrijk om foutmeldingen voor de gebruiker te beperken?**
    *Om te vermijden dat gevoelige informatie (zoals databasefouten) wordt blootgesteld.*

28. **Waarom controleren jullie of `$_POST['username']` en `$_POST['password']` zijn ingevuld?**
    *Om incomplete of foute aanvragen meteen af te wijzen.*

29. **Waarom slaan jullie geen wachtwoorden gehasht op?**
    *Dat zou eigenlijk wél moeten voor veiligheid — dit is een punt van verbetering.*

30. **Wat zou je doen om wachtwoorden veiliger op te slaan?**
    *Gebruik maken van `password_hash()` en `password_verify()` in PHP.*

---

## 🧪 Testen & Validatie

31. **Hoe kun je jullie REST API testen?**
    *Met tools zoals Postman of curl, door POST-verzoeken te sturen.*

32. **Hoe controleer je of je webserver bereikbaar is via HTTPS?**
    *Door naar `https://localhost` of het publieke domein te gaan en te kijken naar het certificaat.*

33. **Hoe test je of de databaseconnectie werkt?**
    *Via de webpagina of rechtstreeks via een psql-verbinding.*

34. **Wat zou je controleren als er geen sensor data zichtbaar is?**
    *Databaseverbinding, queryresultaat, of er überhaupt data in de tabel zit.*

35. **Wat doe je als `php-fpm` niet draait?**
    *Herstarten via `systemctl restart php8.3-fpm`.*

---

## 🔧 Versiebeheer met Git & GitHub

36. **Hoe gebruikten jullie Git in dit project?**
    *Voor versiebeheer, samenwerking en documentatie via commit messages.*

37. **Waarom is `.gitignore` belangrijk?**
    *Het voorkomt dat onnodige bestanden (zoals node_modules, configbestanden) mee in Git komen.*

38. **Wat is een goede commit message?**
    *Kort, beschrijvend, en in de gebiedende wijs, bv. "Fix login bug".*

39. **Hoe toont jullie project samenwerking via GitHub?**
    *Meerdere contributors, regelmatige commits, duidelijke commitgeschiedenis.*

40. **Wat is het nut van `package.json` als je PHP gebruikt?**
    *Waarschijnlijk voor frontend dependencies of tools zoals ESLint, Prettier of bundlers.*

---

## 📟 Embedded Devices & Externe Connectie

41. **Hoe uploadt een embedded device data naar jullie server?**
    *Via een POST-verzoek naar een REST-endpoint.*

42. **Mag het embedded device op dezelfde machine draaien als de server?**
    *Nee, het moet extern zijn om realistische netwerkintegratie te simuleren.*

43. **Hoe kan een ESP32 of Raspberry Pi verbinding maken met jullie server?**
    *Via een HTTP POST-request naar het ingestelde IP-adres of domein van de webserver.*

44. **Wat voor soort data zou het device moeten uploaden?**
    *Temperatuur, vochtigheid, tijdstempel (zoals gesimuleerd in jullie project).*

45. **Hoe zou je authenticatie toevoegen voor het device?**
    *Via een API-key of bearer token.*

---

## 📄 Documentatie & Presentatie

46. **Wat moet er in de README.md staan?**
    *Installatie-instructies, gebruik, uitleg van de structuur en eventuele API-documentatie.*

47. **Waarom is documentatie belangrijk in een projectteam?**
    *Zodat teamleden elkaars werk begrijpen en nieuwe ontwikkelaars kunnen instappen.*

48. **Wat zijn best practices voor het opzetten van een presentatie?**
    *Duidelijke structuur, demo van de werking, uitleg per onderdeel, voorbereiding op vragen.*

49. **Wat zou je tonen in de presentatie als hoogtepunt van het project?**
    *De live sensorgegevens, de REST API werking, de countdown/countup interface.*

50. **Welke leerpunten heb je uit dit project gehaald?**
    *Serverbeheer, beveiliging, PHP, REST APIs, samenwerken via Git, deployment, debugging.*