StudyBuddy handjevastleiding

Hieronder de volledige code, met uitwerkingen en uitleg bij functies. Mocht je kiezen om een andere manier te pakken om de uitwerking te realiseren, ben je hier uiteraard vrij in, mits je het script zelf aan kan passen en de gekozen functies kan toelichten.

Zelfde criteria zijn op het onderstaande ook van toepassing, echter is hier een uitgebreide vooropzet voor neergezet die je kunt bestuderen, gebruiken en uiteraard toepassen.

In onderstaande wordt uitgegaan dat je het volgende al hebt en kan toepassen:

- Visual paradigm → Klassendiagram lezen en maken en ERD lezen en maken.
- Wamp64 → Gebruik van een lokale host met apache, MySQL & PhP 8.x.x

Opdracht - StudyBuddy

Klantverhaal – StudyBuddy

Achtergrond

Wij zijn de studentenraad en het studiecoach-team van **ROC Nova**.
Dagelijks spreken wij veel studenten en merken we dat zij moeite hebben met het organiseren van hun studie. Studenten vergeten deadlines, raken het overzicht kwijt in hun opdrachten en vinden het lastig om samen studieafspraken te plannen. Hierdoor lopen sommige studenten achter of ervaren zij onnodige stress.

Om dit probleem aan te pakken willen wij een eenvoudige webapplicatie laten ontwikkelen met de naam **StudyBuddy**.

StudyBuddy moet een hulpmiddel worden voor studenten om hun studie beter te plannen en samen te werken met klasgenoten. De applicatie moet overzichtelijk, gebruiksvriendelijk en vooral praktisch zijn.

Wat verwachten wij van StudyBuddy?

Accounts en beveiliging

Allereerst willen we dat studenten een eigen account kunnen aanmaken.
Met dit account kunnen zij veilig inloggen en hun persoonlijke gegevens bekijken.
Beveiliging vinden wij belangrijk: wachtwoorden mogen daarom **niet leesbaar** in de database worden opgeslagen.

Taken (persoonlijk overzicht)

Een belangrijk onderdeel van StudyBuddy is het persoonlijke takenoverzicht.

Studenten moeten taken kunnen aanmaken voor hun huiswerk en opdrachten.
Bij elke taak moet een student de volgende gegevens kunnen instellen:

- titel
- beschrijving
- deadline
- prioriteit
- status

Zo kan een student zien wat hij nog moet doen, waar hij mee bezig is en wat al afgerond is.
Studenten moeten hun taken kunnen aanpassen, verwijderen en filteren, bijvoorbeeld door alleen taken te bekijken die nog niet klaar zijn.

Groepen

Daarnaast willen wij dat StudyBuddy samenwerking stimuleert.

Studenten moeten studiegroepen kunnen aanmaken, bijvoorbeeld voor een vak of project.
De student die een groep aanmaakt, wordt automatisch de eigenaar van die groep.

Andere studenten moeten zich bij een groep kunnen aansluiten, bijvoorbeeld via een eenvoudige uitnodiging of code.

Binnen een groep moet zichtbaar zijn wie er lid zijn.

Afspraken binnen groepen

Binnen studiegroepen willen wij dat studenten afspraken kunnen plannen.
Denk hierbij aan samen studeren, werken aan een project of een online meeting.

Bij elke afspraak moet worden vastgelegd:

- datum en tijd
- locatie of online-link
- onderwerp

Alleen leden van de groep mogen deze afspraken zien.
Per afspraak moet elke student kunnen aangeven of hij:

- erbij is
- misschien komt
- niet kan

Dashboard en overzicht

Om alles overzichtelijk te houden, willen wij een dashboard.

Op dit dashboard ziet een student in één oogopslag:

- het aantal openstaande taken
- het percentage afgeronde taken
- de eerstvolgende afspraak (als die bestaat)

Binnen een groep willen we kunnen zien:

- hoeveel leden de groep heeft
- welke afspraken er binnenkort gepland staan

Gebruikers van het systeem

Wij onderscheiden twee soorten gebruikers:

- **Studenten**
  Gebruiken de applicatie om taken te plannen, groepen te maken en afspraken te beheren.
- **Admins** (bijvoorbeeld studiecoaches)
  Kunnen gebruikers beheren als dat nodig is, zoals het blokkeren of verwijderen van accounts of het aanpassen van rollen.

Technische verwachtingen

Wij verwachten dat de applicatie gebouwd wordt met **PHP volgens Object Oriented Programming (OOP)**.

- Database: **MySQL**
- Data ophalen en opslaan met **PDO en prepared statements**

Voordat er begonnen wordt met programmeren, willen wij eerst een duidelijk ontwerp zien. Dit ontwerp moet bestaan uit:

- minimaal **10 user stories** met acceptatiecriteria;
- een **ERD** met minimaal 6 tabellen en duidelijke relaties;
- een **klassendiagram** met minimaal 6 PHP-classes, inclusief properties en methodes.

Eindresultaat

Aan het einde van het project willen wij:

1. Een werkende webapplicatie met de belangrijkste functionaliteiten van StudyBuddy.
2. Een duidelijk ontwerpdocument met user stories, ERD en klassendiagram.
3. Een korte README waarin staat hoe de applicatie geïnstalleerd kan worden en hoe wij kunnen inloggen met een testaccount.

Beoordeling

Deze opdracht wordt beoordeeld op 2 fronten.

1. Een werkende webapplicatie door middel van object oriënted programming in PHP en MySQL --> Beoordeeld met een code review.
2. Een criterium gericht interview bij oplevering van het project.

Hoe aangetoond dat dit is afgerond

1. Deelnemer ontvangt een certificaat voor gevorderd object oriënted programming nv4 voor in het leerportfolio.

Scope en kernfunctionaliteiten

StudyBuddy helpt studenten met:

1. **Accounts & beveiliging** (registreren, inloggen, wachtwoorden gehasht).
2. **Persoonlijke taken** (CRUD + filteren + status/prioriteit).
3. **Studiegroepen** (aanmaken, joinen via code, ledenlijst).
4. **Afspraken in groepen** (plannen, alleen zichtbaar voor leden).
5. **Aanwezigheid per afspraak** (erbij/misschien/niet).
6. **Dashboard** (open taken, % afgerond, eerstvolgende afspraak; en binnen groep: leden + komende afspraken).
7. **Admin-rollen** (gebruikers/rollen beheren).

User stories met acceptatiecriteria

US01 — Registreren

Als student wil ik een account kunnen aanmaken zodat ik StudyBuddy persoonlijk kan gebruiken.
Acceptatiecriteria

- Gegeven dat ik naam, e-mail en wachtwoord invul, wanneer ik registreer, dan wordt er een nieuwe gebruiker aangemaakt.
- E-mail is uniek; bij bestaande e-mail krijg ik een duidelijke foutmelding.
- Wachtwoord wordt **niet** leesbaar opgeslagen maar als **hash** (bijv. password_hash).

US02 — Inloggen/uitloggen

Als student wil ik kunnen inloggen en uitloggen zodat mijn gegevens beschermd zijn.
Acceptatiecriteria

- Wanneer ik een correct e-mail+wachtwoord invoer, dan krijg ik toegang tot mijn dashboard.
- Bij een fout wachtwoord of onbekende e-mail krijg ik “inloggegevens onjuist”.
- Uitloggen maakt de sessie ongeldig en brengt me terug naar login.

US03 — Profiel bekijken

Als student wil ik mijn basisgegevens kunnen bekijken zodat ik weet welk account ik gebruik.
Acceptatiecriteria

- Dashboard of profielpagina toont minimaal: naam, e-mail, rol.
- Gegevens komen uit users tabel op basis van sessie-user.

US04 — Taak aanmaken

Als student wil ik een taak kunnen toevoegen zodat ik deadlines en opdrachten bijhoud.
Acceptatiecriteria

- Taak bevat: titel (verplicht), beschrijving (optioneel), deadline (optioneel), prioriteit, status.
- Na opslaan verschijnt de taak in mijn takenlijst.
- Taak is gekoppeld aan mijn userid.

US05 — Taken beheren (wijzigen/verwijderen)

Als student wil ik taken kunnen aanpassen of verwijderen zodat mijn overzicht actueel blijft.
Acceptatiecriteria

- Alleen de eigenaar (zelfde userid) kan wijzigen/verwijderen.
- Wijzigingen worden direct zichtbaar in het overzicht.
- Verwijderen vraagt bevestiging en verwijdert record uit tasks.

US06 — Taken filteren op status

Als student wil ik mijn taken kunnen filteren (bijv. alleen open) zodat ik focus houd.
Acceptatiecriteria

- Filter “Open” toont alleen taken waarvan status ≠ “done”.
- Filter werkt server-side (PDO prepared statement) en/of client-side, maar data is correct.
- “Alles” toont alle taken van de gebruiker.

US07 — Studiegroep aanmaken (owner)

Als student wil ik een studiegroep kunnen maken zodat ik met anderen kan samenwerken.
Acceptatiecriteria

- Groep heeft: naam (verplicht), beschrijving (optioneel), join_code (uniek).
- Maker wordt eigenaar (in ERD: groups.userid verwijst naar maker; in klassendiagram: ownerId).
- Maker wordt tevens lid (record in users_groups).

US08 — Lid worden via join-code

Als student wil ik via een code lid kunnen worden zodat ik eenvoudig kan aansluiten.
Acceptatiecriteria

- Wanneer ik een geldige join_code invoer, dan word ik toegevoegd aan users_groups.
- Ongeldige code geeft foutmelding.
- Als ik al lid ben, wordt geen duplicaat aangemaakt.

US09 — Groepsleden bekijken

Als student wil ik zien wie er in mijn groep zitten zodat ik afspraken kan maken.
Acceptatiecriteria

- Groepspagina toont ledenlijst (via users_groups → users).
- Alleen leden kunnen deze pagina zien; niet-leden krijgen “geen toegang”.

US10 — Afspraak plannen in groep

Als student wil ik een afspraak in een groep kunnen plannen zodat we samen kunnen studeren.
Acceptatiecriteria

- Afspraak bevat: datum/tijd, locatie/online-link, onderwerp.
- Afspraak is gekoppeld aan groupsid en maker userid (ERD: meetings.groupsid, meetings.userid).
- Alleen groepsleden kunnen de afspraken zien.

US11 — Aanwezigheid per afspraak doorgeven

Als student wil ik per afspraak aangeven of ik erbij ben zodat de groep weet wie komt.
Acceptatiecriteria

- Statusopties: “erbij”, “misschien”, “niet”.
- Opslag in koppel-tabel users_meetings met statusid en updated_at.
- Alleen leden van de groep kunnen hun status zetten.

US12 — Dashboard met kerncijfers

Als student wil ik een dashboard met statistieken zodat ik snel overzicht heb.
Acceptatiecriteria

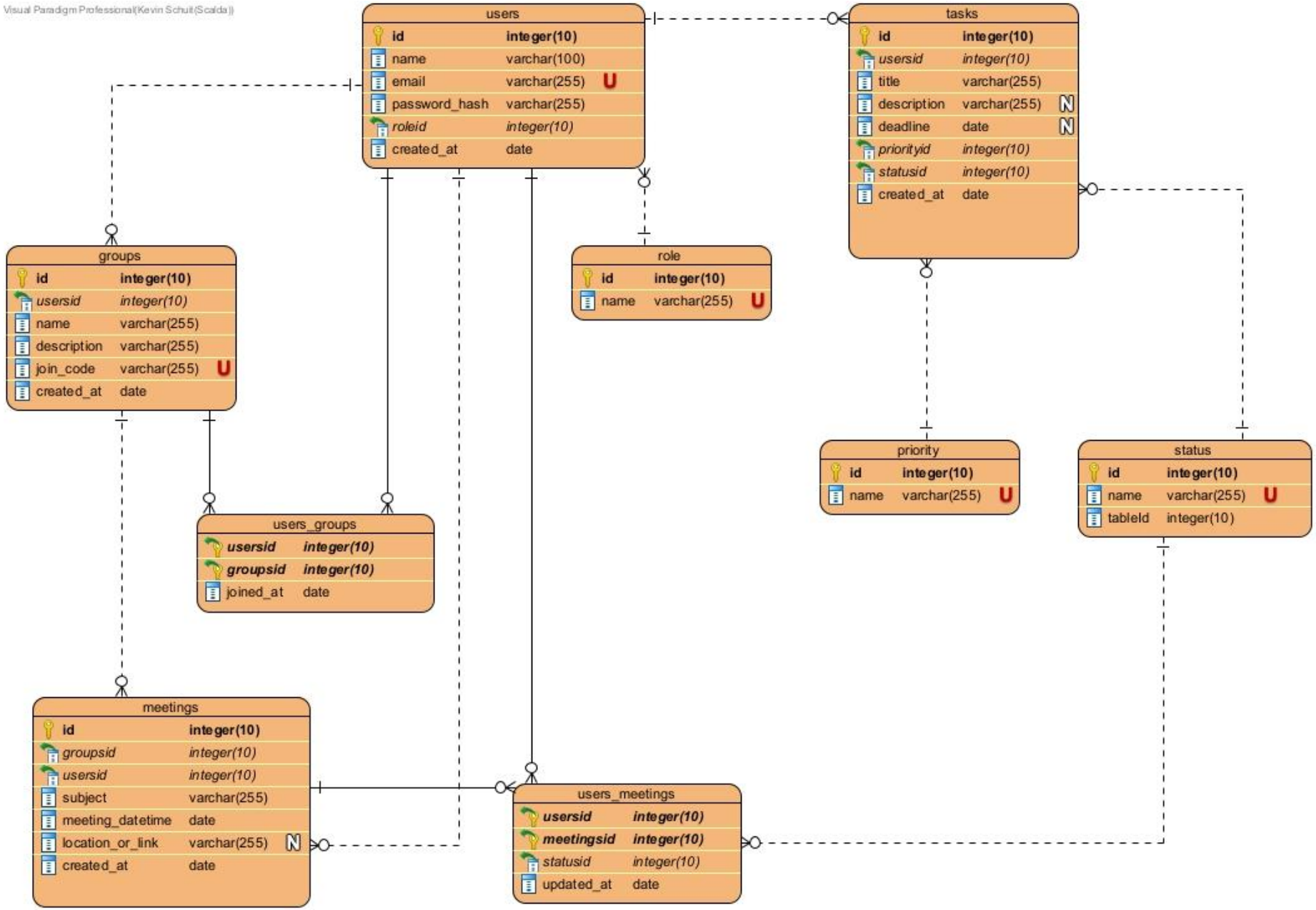
- Dashboard toont:
    - aantal open taken (count)
    - percentage afgerond (done / totaal * 100)
    - eerstvolgende afspraak (min datum/tijd vanaf nu) of “geen afspraak”
- Alleen data van ingelogde gebruiker.

US13 — Admin: gebruiker beheren (blokkeer/verwijder/rol)

Als admin wil ik gebruikers kunnen beheren zodat misbruik aangepakt kan worden.
Acceptatiecriteria

- Alleen gebruikers met rol “admin” zien beheerfuncties.
- Admin kan gebruiker verwijderen (cascade/veilig verwijderen) of blokkeren (als je “blocked” toevoegt: extra kolom of status).
- Admin kan rol aanpassen via users.roleid.

ERD (MySQL) — tabellen en relaties



Sql/tables.sql

```sql
CREATE DATABASE studybuddy;
-- USE studybuddy;

SET FOREIGN_KEY_CHECKS = 0;

DROP TABLE IF EXISTS users_meetings;
DROP TABLE IF EXISTS users_groups;
DROP TABLE IF EXISTS meetings;
DROP TABLE IF EXISTS tasks;
DROP TABLE IF EXISTS `groups`;
DROP TABLE IF EXISTS users;
DROP TABLE IF EXISTS status;
DROP TABLE IF EXISTS priority;
DROP TABLE IF EXISTS role;

SET FOREIGN_KEY_CHECKS = 1;

-- 1) ROLE
CREATE TABLE role (
  id INT(10) NOT NULL AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY uq_role_name (name)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 2) USERS
CREATE TABLE users (
  id INT(10) NOT NULL AUTO_INCREMENT,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(255) NOT NULL,
```

```
  password_hash VARCHAR(255) NOT NULL,
  roleid INT(10) NOT NULL,
  created_at DATE NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY uq_users_email (email),
  KEY idx_users_roleid (roleid),
  CONSTRAINT fk_users_role
    FOREIGN KEY (roleid) REFERENCES role(id)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 3) PRIORITY
CREATE TABLE priority (
  id INT(10) NOT NULL AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY uq_priority_name (name)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 4) STATUS
CREATE TABLE status (
  id INT(10) NOT NULL AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  tableId INT(10) NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY uq_status_name (name)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 5) GROUPS
CREATE TABLE `groups` (
```

```
  id INT(10) NOT NULL AUTO_INCREMENT,
  userid INT(10) NOT NULL,
  name VARCHAR(255) NOT NULL,
  description VARCHAR(255) NOT NULL,
  join_code VARCHAR(255) NOT NULL,
  created_at DATE NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY uq_groups_join_code (join_code),
  KEY idx_groups_userid (userid),
  CONSTRAINT fk_groups_user
    FOREIGN KEY (userid) REFERENCES users(id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
)) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 6) USERS_GROUPS (koppeltabel)
CREATE TABLE users_groups (
  userid INT(10) NOT NULL,
  groupsid INT(10) NOT NULL,
  joined_at DATE NOT NULL,
  PRIMARY KEY (userid, groupsid),
  KEY idx_users_groups_groupsid (groupsid),
  CONSTRAINT fk_users_groups_user
    FOREIGN KEY (userid) REFERENCES users(id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk_users_groups_group
    FOREIGN KEY (groupsid) REFERENCES `groups`(id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
)) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
93)
94)-- 7) TASKS
95)CREATE TABLE tasks (
96)  id INT(10) NOT NULL AUTO_INCREMENT,
97)  userid INT(10) NOT NULL,
98)  title VARCHAR(255) NOT NULL,
99)  description VARCHAR(255) NULL,
100)       deadline DATE NULL,
101)       priorityid INT(10) NOT NULL,
102)       statusid INT(10) NOT NULL,
103)       created_at DATE NOT NULL,
104)       PRIMARY KEY (id),
105)       KEY idx_tasks_userid (userid),
106)       KEY idx_tasks_priorityid (priorityid),
107)       KEY idx_tasks_statusid (statusid),
108)       CONSTRAINT fk_tasks_user
109)         FOREIGN KEY (userid) REFERENCES users(id)
110)         ON DELETE CASCADE
111)         ON UPDATE CASCADE,
112)       CONSTRAINT fk_tasks_priority
113)         FOREIGN KEY (priorityid) REFERENCES priority(id)
114)         ON DELETE RESTRICT
115)         ON UPDATE CASCADE,
116)       CONSTRAINT fk_tasks_status
117)         FOREIGN KEY (statusid) REFERENCES status(id)
118)         ON DELETE RESTRICT
119)         ON UPDATE CASCADE
120)     ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
121)
122)     -- 8) MEETINGS
123)     CREATE TABLE meetings (
```

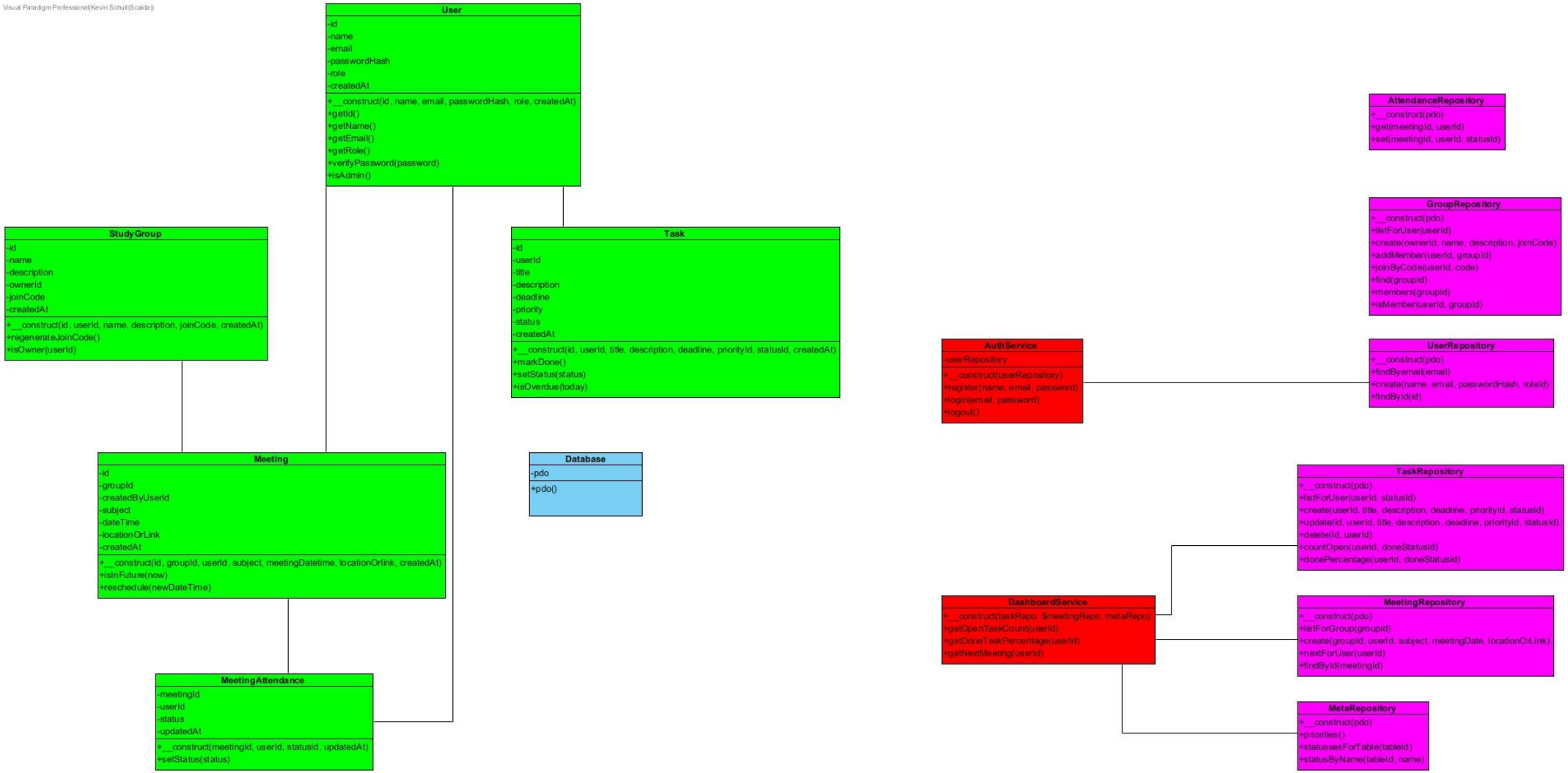
```
124)        id INT(10) NOT NULL AUTO_INCREMENT,
125)        groupsid INT(10) NOT NULL,
126)        userid INT(10) NOT NULL,
127)        subject VARCHAR(255) NOT NULL,
128)        meeting_datetime DATE NOT NULL,
129)        location_or_link VARCHAR(255) NULL,
130)        created_at DATE NOT NULL,
131)        PRIMARY KEY (id),
132)        KEY idx_meetings_groupsid (groupsid),
133)        KEY idx_meetings_userid (userid),
134)        CONSTRAINT fk_meetings_group
135)          FOREIGN KEY (groupsid) REFERENCES `groups`(id)
136)          ON DELETE CASCADE
137)          ON UPDATE CASCADE,
138)        CONSTRAINT fk_meetings_user
139)          FOREIGN KEY (userid) REFERENCES users(id)
140)          ON DELETE CASCADE
141)          ON UPDATE CASCADE
142)      ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
143)
144)      -- 9) USERS_MEETINGS (koppeltabel + status)
145)      CREATE TABLE users_meetings (
146)        userid INT(10) NOT NULL,
147)        meetingsid INT(10) NOT NULL,
148)        statusid INT(10) NOT NULL,
149)        updated_at DATE NOT NULL,
150)        PRIMARY KEY (userid, meetingsid),
151)        KEY idx_users_meetings_meetingsid (meetingsid),
152)        KEY idx_users_meetings_statusid (statusid),
153)        CONSTRAINT fk_users_meetings_user
154)          FOREIGN KEY (userid) REFERENCES users(id)
```

```
155)         ON DELETE CASCADE
156)         ON UPDATE CASCADE,
157)       CONSTRAINT fk_users_meetings_meeting
158)         FOREIGN KEY (meetingsid) REFERENCES meetings(id)
159)         ON DELETE CASCADE
160)         ON UPDATE CASCADE,
161)       CONSTRAINT fk_users_meetings_status
162)         FOREIGN KEY (statusid) REFERENCES status(id)
163)         ON DELETE RESTRICT
164)         ON UPDATE CASCADE
165)     ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Sql/seed.sql

```sql
INSERT INTO role (name) VALUES ('student'), ('admin');

INSERT INTO priority (name) VALUES ('laag'), ('middel'), ('hoog');

-- tableId 1 = tasks
INSERT INTO status (name, tableId) VALUES
('todo', 1), ('doing', 1), ('done', 1);

-- tableId 2 = meetings (attendance)
INSERT INTO status (name, tableId) VALUES
('yes', 2), ('maybe', 2), ('no', 2);

-- Testaccount handig voor oplevering
-- Username: test@student.nl
-- Password: password
INSERT INTO users (name, email, password_hash, roleid, created_at)
VALUES (
  'Test Student',
  'test@student.nl',
  '$2y$10$eImiTXuWVxfM37uY4JANjQ==',
  1,
  CURDATE()
);
```

Klassendiagram (PHP OOP) — classes, properties, methodes



Technische uitwerking (PHP + PDO) — ontwerpkeuzes

Security

- Wachtwoorden: password_hash() + password_verify()
- Prepared statements overal
- Sessies met server-side session id
- Autorisatie checks:
    - Alleen eigenaar mag taak wijzigen/verwijderen
    - Alleen groepsleden mogen meetings zien
    - Alleen admin mag user management

Logica vanuit ERD/klassendiagram

- Task status/prioriteit via MetaRepository → dropdowns in UI
- Attendance status via users_meetings.statusid (met status table die ook voor tasks gebruikt kan worden via tableId)
- Dashboard:
    - TaskRepository::countOpen(userId, doneStatusId)
    - TaskRepository::donePercentage(userId, doneStatusId)
    - MeetingRepository::nextForUser(userId) (meetings via groups waar user lid is)

README (installatie + testaccount)

Vereisten

- PHP 8.x
- MySQL 5.7+ / 8.x
- Webserver (Apache/Nginx) of PHP built-in server
- Composer (optioneel, als je autoloading gebruikt)

Installatie

1. Clone/download het project naar je webserver-map.
2. Maak een MySQL database aan, bijv. studybuddy.
3. Importeer het SQL-schema (tabellen uit ERD).
4. Zet database credentials in config.php (of .env):
    - DB_HOST, DB_NAME, DB_USER, DB_PASS
5. Start de applicatie:
    - via Apache/Nginx, of:
    - php -S localhost:8000 -t public

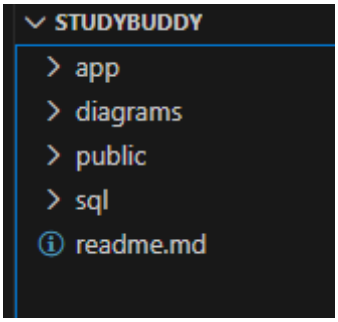
Inloggen met testaccounts

- **Student**
    - Email: student@test.nl
    - Wachtwoord: Test123!
- **Admin**
    - Email: admin@test.nl
    - Wachtwoord: Admin123!

Kernfunctionaliteiten om te testen

- Registreren/inloggen
- Taken aanmaken + status “done” zetten + filter “open”
- Groep maken → join code kopiëren → tweede gebruiker joinen
- Afspraak plannen in groep → aanwezigheid “misschien” instellen
- Dashboard: open taken / % done / eerstvolgende afspraak

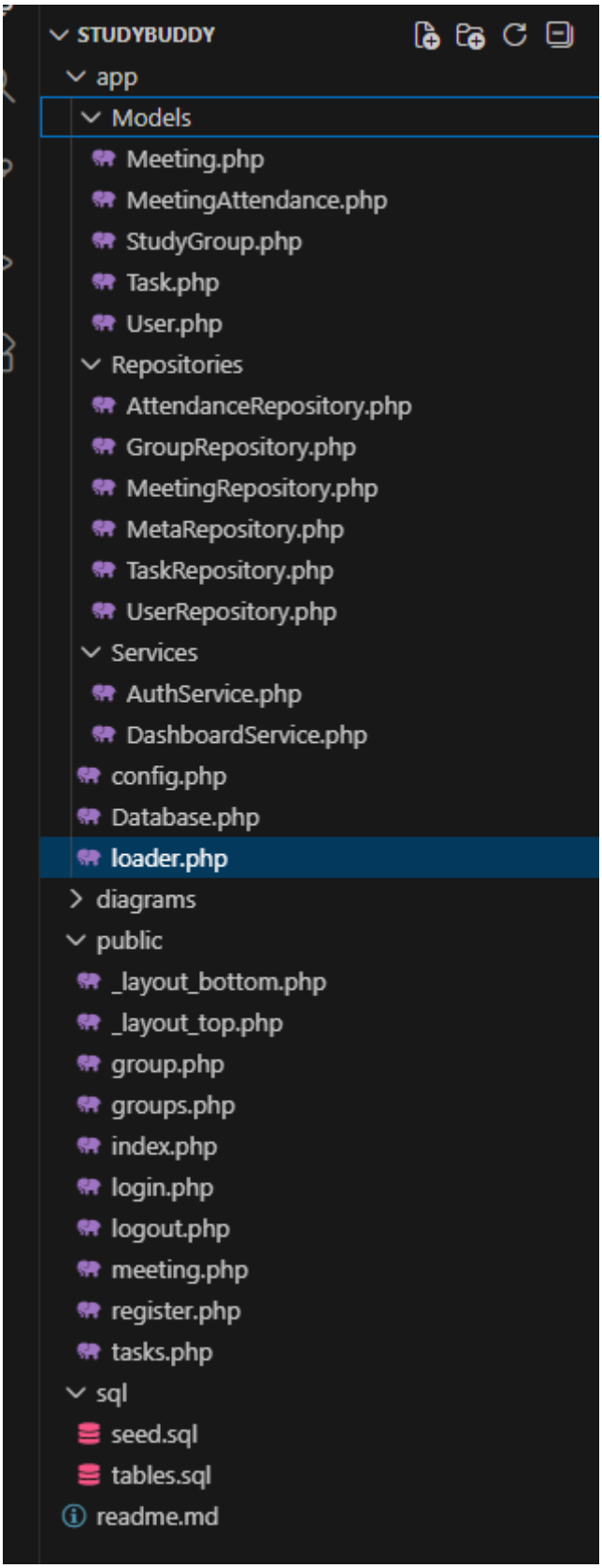
Projectstructuur



Waarom zo?

- public/ = pagina's die je opent in de browser.
- app/ = alle classes (OOP).
- diagrams/ = diagrammen van visual paradigm.
- sql/ = database scripts.

Maak de volgende mappenstructuur:



Je bestaande klassen kun je in app/Models en app/Services plaatsen.

De logica binnen de app/ map is als volgt:

Bestand/map	Eigenschap
database.php	bevat de database connectie.
loader.php	bevat de autoloader, current_user check en redirection.
config.php	bevat de database configuratie.
Models/	Bevat de modellen/ checks/ getters en setters
Repositories/	Bevat functies die de database verwerking uitvoeren
Services/	Bevat services die dashboard checks en login/register en logout checks uitvoeren buiten de normale klasses.

config + Database connectie

Zorg dat je in phpmyadmin een database hebt die studdybuddy heet.

Dit kun je doen door naar de volgende link te gaan (mits je wamp64 aan hebt staan).

http://localhost/phpmyadmin/

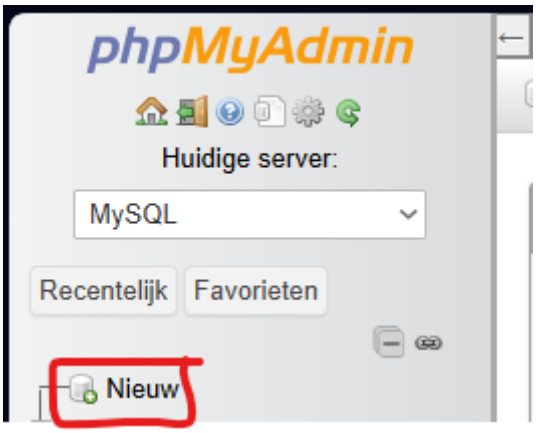
Je komt dan hier

Je logt in met :

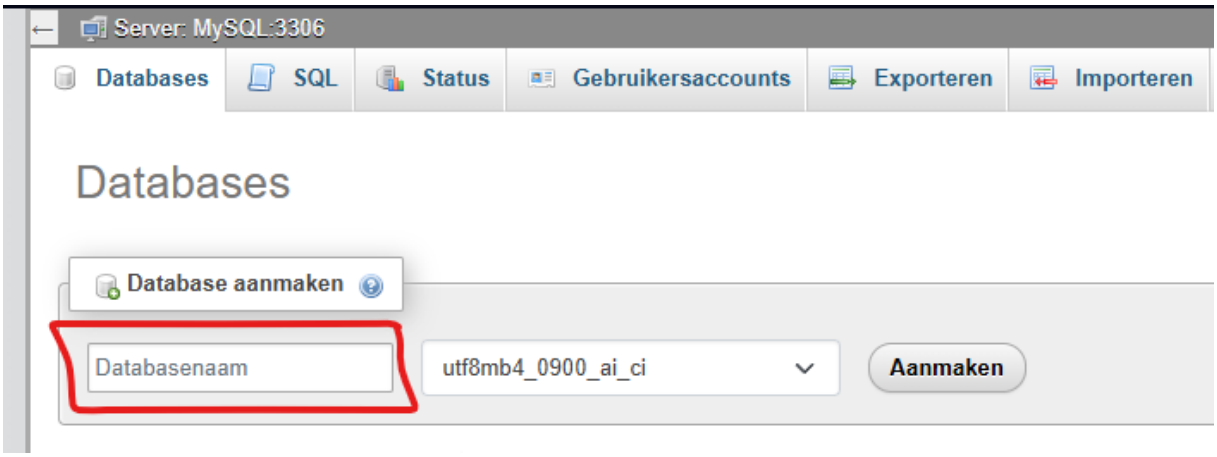
Gebruikersnaam: root

Wachtwoord:

Klik nu op: Nieuw



Voer de naam voor je database in en klik op aanmaken:



In app/config.php zet je het volgende:

Zorg wel dat ‘name’ de naam van je database is, als deze geen ‘studdybuddy’ is, pas de naam dan zodanig aan. De andere gegeven kun je zo laten.

Voer de sql queries in Sql/tables.sql en Sql/seed.sql uit door op de database te klikken en vervolgens op ‘SQL’ en de code hierin uit te voeren.

De karakterset (charset) kun je uit de 2e kolom van de bovenstaande afbeelding halen.

app > config.php

```php
<?php
return [
    'db' => [
        'host' => 'localhost',
        'name' => 'studybuddy',
        'user' => 'root',
        'pass' => '',
        'charset' => 'utf8mb4',
    ]
];
```

Zet in het bestand app/Database.php de volgende code:

Database.php

```php
<?php
class Database {
  private static ?PDO $pdo = null;

  public static function pdo(): PDO {
    if (self::$pdo) return self::$pdo;

    $config = require __DIR__ . '/config.php';
    $db = $config['db'];

    $dsn = "mysql:host={$db['host']};dbname={$db['name']};charset={$db['charset']}";
    self::$pdo = new PDO($dsn, $db['user'], $db['pass'], [
      PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
      PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
      PDO::ATTR_EMULATE_PREPARES => false
    ]);
    return self::$pdo;
  }
}
```

Wat hebben we gedaan:

We hebben de klasse “Database” aangemaakt

Vervolgens hebben we een leeg PDO object aangemaakt.

- $pdo is **static** → er is **één gedeelde databaseverbinding** voor de hele applicatie
- null betekent: nog geen verbinding gemaakt
- ?PDO = kan een PDO object zijn óf null

public static function pdo(): PDO {

- Deze methode geeft altijd een **PDO-object** terug
- Je roept haar aan met: Database::pdo()

if (self::$pdo) return self::$pdo;

Als de verbinding al bestaat → **hergebruik hem**

Geen nieuwe connectie = sneller & efficiënter

$config = require __DIR__ . '/config.php';

- $db = $config['db'];
- Laadt config.php

Verwacht iets als:

app > config.php

```php
<?php
return [
    'db' => [
        'host' => 'localhost',
        'name' => 'studybuddy',
        'user' => 'root',
        'pass' => '',
        'charset' => 'utf8mb4',
    ]
];
```

$dsn = "mysql:host={$db['host']};dbname={$db['name']};charset={$db['charset']}";

Dit vertelt PDO:

- welk type database (mysql)
- waar hij staat (host)
- welke database (dbname)
- welke tekenset (charset)

Hier maak je de echte verbinding met de database.

Vervolgens check je of er een error ontstaat en zet je exception meldingen door.

Je kijkt ook of er assosciatieve arrays zijn zoals: ['id' => 1, 'naam' => 'Jan']

Als laatst geef je aan dat je geen prepared statement mag emuleren, wat veiliger is tegen SQL injectie.

```php
self::$pdo = new PDO($dsn, $db['user'], $db['pass'], [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES => false
]);
```

Als laatste return self::$pdo;

Nu heeft de gehele applicatie waar deze klasse gebruikt wordt, dezelfde verbinding met de database.

Loader

Wat wij nu gaan doen is de loader.php inrichten. Dit bestand is naast de database één van de meer cruciale bestanden.

De app/loader.php ziet er als volgt uit:

Wat hier gebeurt is redelijk eenvoudig.

Session_start(); start een browser sessie, waarin je tijdens de applicatie dingen op kunt slaan en heropenen, zoals bijvoorbeeld jouw profiel naam, paginatie en meer.

Vervolgens koppelen wij de database doormiddel van een require_once.

Je hebt naast de require_once ook een include_once.

Verschil tussen require en include, is dat je bij require het bestand nodig hebt om iets te doen.

De _once geeft aan dat het bestand maar één keer geopend kan worden, dit voorkomt dubbele instanties binnen bestanden.

__DIR__ geeft aan dat we het directory pad (map pad) willen benaderen waar het bestand zich bevind, waar dit in getypt wordt.

De sql_autoload_register functie is een array waarin wij alle bestanden binnen de Models, Services, Repositories en huidige map aan kunnen roepen, zonder ze allemaal uit te hoeven schrijven per require_once.

```php
spl_autoload_register(function ($class) {
  $paths = [
    __DIR__ . "/Models/{$class}.php",
    __DIR__ . "/Services/{$class}.php",
    __DIR__ . "/Repositories/{$class}.php",
    __DIR__ . "/{$class}.php",
  ];
  foreach ($paths as $p) {
    if (file_exists($p)) { require_once $p; return; }
  }
});
```

Deze functie zorgt er voor dat alle data uit de database omgezet kan worden naar UTF-8 (wat html bestanden lezen). Data uit een database kan soms onbedoeld omgezet worden naar andere karakters. Dit onderschept dat probleem.

```php
function e(string $s): string {
  return htmlspecialchars($s, ENT_QUOTES, 'UTF-8');
}
```

Deze functie stuurt ons terug naar de pagina waar we vandaan komen, dit gebruiken we later bij het verzenden van data naar de database. Normaliter kun je de pagina verversen en blijf je de zelfde data opnieuw sturen naar de database, wat er voor kan zorgen dat je veel dezelfde data hebt.

```php
function redirect(string $to): void {
  header("Location: {$to}");
  exit;
}
```

Deze functie haalt de huidige gebruiker op uit de sessie. Hiervoor gebruiken we dus session_start();

```php
function current_user(): ?array {
  return $_SESSION['user'] ?? null;
}
```

Met deze functie kijken we of de gebruiker is ingelogt, zo niet, dan sturen we de gebruiker naar de login pagina.

We kijken hier of de gebruiker NIET de current_user() is, door deze functie eerst aan te roepen. Hierna, roepen we de redirect aan en sturen we de gebruiker naar de login.php pagina.

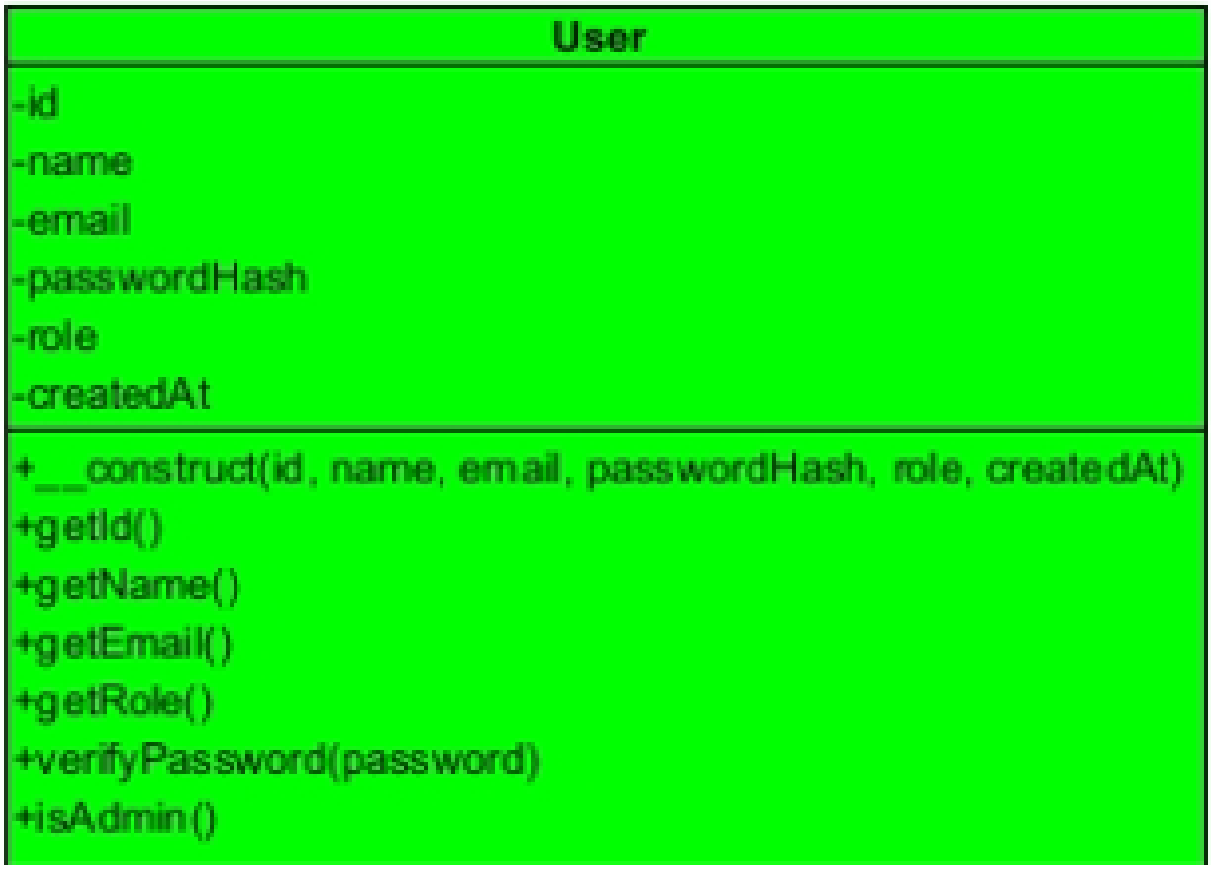
```php
function require_login(): void {
  if (!current_user()) redirect('login.php');
}
```

Models/

Nu gaan we de app/Models/User.php aanmaken

De andere modellen zijn redelijk hetzelfde, hiervoor leg ik enkel deze klasse uit.

We zoemen in op de volgende klassen:



Een klasse zit als volgt in elkaar.

- Klasse naam
- Attributen
- Operators (functies)

De attributen kun je veelal uit een vraag of ontwerp halen, deze komen ook overeen met je database ontwerp. Een attribuut is 9/10 keer een private variabele. Je hebt de volgende mogelijkheden

Operator	Operator uitleg
-	Private operator, deze kun je enkel binnen de eigen klasse gebruiken
+	Public operator, deze kun je zowel binnen als buiten de klasse gebruiken.
#	Protected operator, deze kun je enkel binnen de klasse gebruiken of klasses die afkomstig zijn uit deze klasse.

In de klasse hebben we de volgende attributen aangemaakt:

Id, name, email, passwordHash, role, createdAt

Al deze atributen zijn private, omdat we ze niet buiten de klasse kunnen en mogen instnatiëren, verder hebben we ze ook niet nodig in een lagere klasse, sinds dit de laagste is.

Vervolgens hebben we een aantal functies. Deze functies worden buiten de klasse aangeroepen en gebruikt, bijvoorbeeld in de register.php die we later aanmaken. Hierdoor moeten deze functies public zijn.

Ook 9/10 functies zijn altijd public, dus dit is een eenvoudige reminder.

We hebben minimaal de volgende getters nodig:

getId, getName, getEmail, getRole.

```php
public function __construct($id, $name, $email, $passwordHash, $role, $createdAt) {
  $this->id = (int)$id;
  $this->name = $name;
  $this->email = $email;
  $this->passwordHash = $passwordHash;
  $this->role = $role;
  $this->createdAt = $createdAt;
}
```

```php
public function getId() { return $this->id; }
public function getName() { return $this->name; }
public function getEmail() { return $this->email; }
public function getRole() { return $this->role; }
```

Dit heeft de volgende reden: we hoeven niet altijd te zien wanneer een user is aangemaakt, sinds we geen user overzicht hebben, daarnaast wil je ook geen wachtwoord op kunnen roepen, enkel vergelijken bij inloggen. Dus deze hoeven we ook niet te zien.

Het Id hebben we nodig om de user te kunnen identificeren.

De name is nodig om de naam te tonen, hetzelfde voor de email en role. De role is hier nog het meest afhankelijk, omdat deze de rechten van de gebruiker gaat bepalen.

```php
public function verifyPassword($password) {
  return password_verify($password, $this->passwordHash);
}
```

Hierna hebben we verifyPassword($password), deze functie benut de verify_password functie van php, die controleert of het ingevoerde wachtwoord ook overeenkomt met onze passwordHash uit onze database.

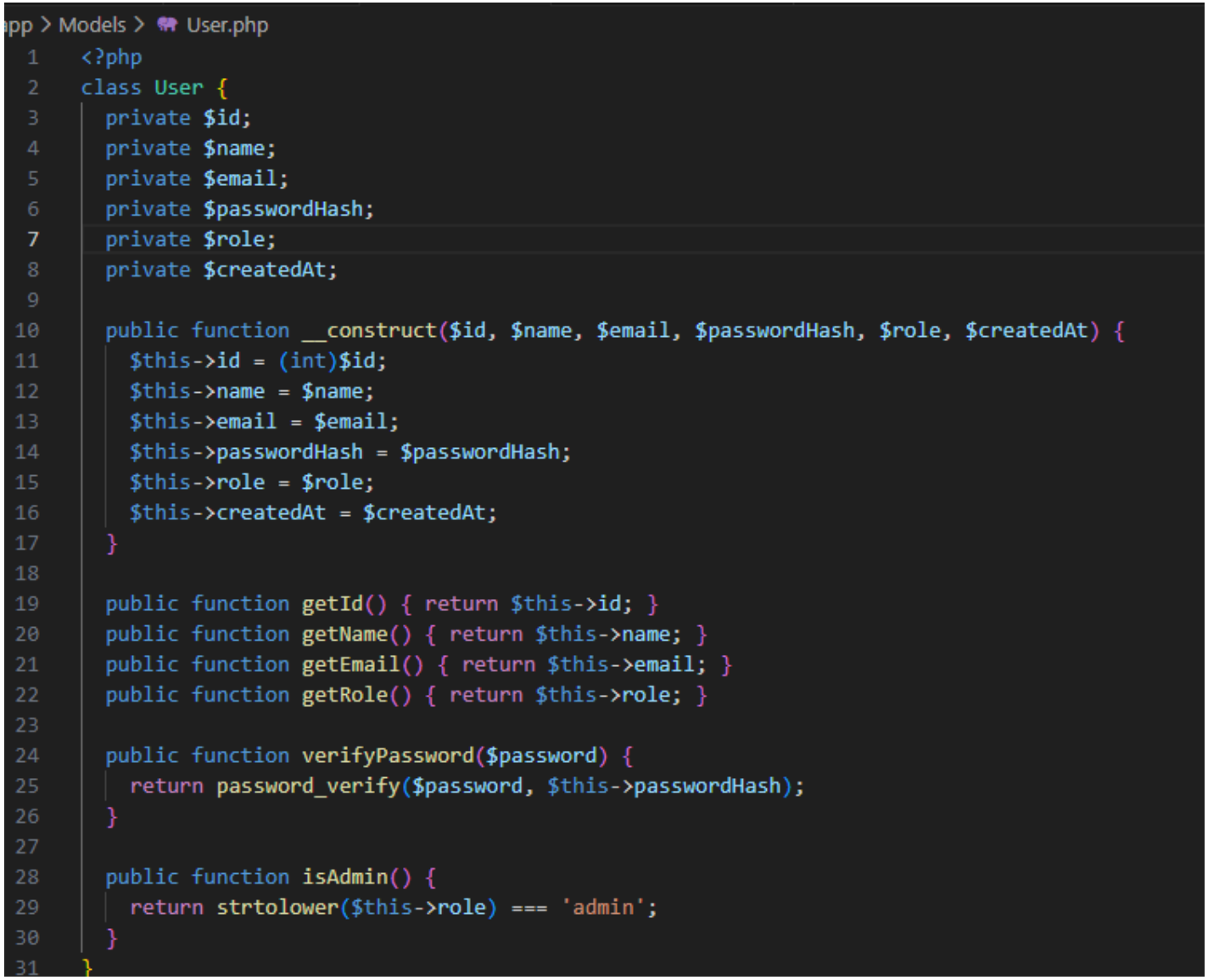
```php
public function isAdmin() {
    return strtolower($this->role) === 'admin';
}
```

De functie isAdmin() bekijkt of de gebruiker de rol van admin heeft, zo niet, dan mag deze waarschijnlijk niet verder. Strtolower($this->role) zet de opgehaalde rol waarde uit de database ook naar lowercase, zodat deze ook volledig overeen komt met ‘admin’.

Stel dat de database nu Admin opgeslagen heeft, dan past strtolower dit dus aan van ‘Admin’ naar ‘admin’.

De klasse ziet er dan dus als volgt uit:

app/Models/User.php



App/Models/Task.php

app > Models > Task.php

```php
<?php
class Task {
    public $id;
    public $userId;
    public $title;
    public $description;
    public $deadline;
    public $priorityId;
    public $statusId;
    public $createdAt;

    public function __construct($id, $userId, $title, $description, $deadline, $priorityId, $statusId, $createdAt) {
        $this->id = (int)$id;
        $this->userId = (int)$userId;
        $this->title = $title;
        $this->description = $description;
        $this->deadline = $deadline;
        $this->priorityId = (int)$priorityId;
        $this->statusId = (int)$statusId;
        $this->createdAt = $createdAt;
    }

    public function markDone() {
        // statusId wordt in repository gezet (done-id)
    }

    public function setStatus($status) {
        $this->statusId = (int)$status;
    }

    public function isOverdue($today) {
        if (!$this->deadline) return false;
        return $this->deadline < $today;
    }
}
```

App/Models/StudyGroup.php

app > Models > StudyGroup.php

```php
<?php
class StudyGroup {
    public $id;
    public $userId;
    public $name;
    public $description;
    public $joinCode;
    public $createdAt;

    public function __construct($id, $userId, $name, $description, $joinCode, $createdAt) {
        $this->id = (int)$id;
        $this->userId = (int)$userId;
        $this->name = $name;
        $this->description = $description;
        $this->joinCode = $joinCode;
        $this->createdAt = $createdAt;
    }

    public function regenerateJoinCode() {
        $this->joinCode = substr(bin2hex(random_bytes(8)), 0, 10);
    }

    public function isOwner($userId) {
        return $this->userId === (int)$userId;
    }
}
```

App/Models/MeetingAttendance.php

app > Models > MeetingAttendance.php

```php
<?php
class MeetingAttendance {
    public $meetingId;
    public $userId;
    public $statusId;
    public $updatedAt;

    public function __construct($meetingId, $userId, $statusId, $updatedAt) {
        $this->meetingId = (int)$meetingId;
        $this->userId = (int)$userId;
        $this->statusId = (int)$statusId;
        $this->updatedAt = $updatedAt;
    }

    public function setStatus($status) {
        $this->statusId = (int)$status;
    }
}

```

App/Models/Meeting.php

app > Models > Meeting.php

```php
<?php
class Meeting {
    public $id;
    public $groupId;
    public $userId;
    public $subject;
    public $meetingDatetime;
    public $locationOrLink;
    public $createdAt;

    public function __construct($id, $groupId, $userId, $subject, $meetingDatetime, $locationOrLink, $createdAt) {
        $this->id = (int)$id;
        $this->groupId = (int)$groupId;
        $this->userId = (int)$userId;
        $this->subject = $subject;
        $this->meetingDatetime = $meetingDatetime;
        $this->locationOrLink = $locationOrLink;
        $this->createdAt = $createdAt;
    }

    public function isInFuture($now) {
        return $this->meetingDatetime >= $now;
    }

    public function reschedule($newDateTime) {
        $this->meetingDatetime = $newDateTime;
    }
}
```

Repositories

Wat doen repository classes?

Een **repository class** is verantwoordelijk voor **alle database-operaties van één onderdeel van de applicatie**.
In dit geval beheert AttendanceRepository de aanwezigheid (*attendance*) van gebruikers bij meetings.

De rest van de applicatie (controllers, services) **schrijft geen SQL**, maar gebruikt alleen de methodes van de repository.

Waarom repositories gebruiken?

- SQL staat op **één centrale plek**
- Code wordt **overzichtelijker**
- Makkelijker te **onderhouden en testen**
- Losgekoppeld van PDO / database-details

De constructor

```
public function __construct(private PDO $pdo) {}
```

De repository krijgt een bestaande databaseverbinding mee (*dependency injection*).
Hij maakt zelf **geen** verbinding aan.

get($meetingId, $userId)

Haalt de attendance-gegevens op van één gebruiker voor één meeting.

- Zoekt in users_meetings
- Haalt ook de naam van de status op (status_name)
- Gebruikt prepared statements (veilig)
- Geeft een **array** terug of **null** als er geen record bestaat

set($meetingId, $userId, $statusId)

Slaat de attendance-status op.

- Doet een **INSERT**
- Bestaat het record al → **UPDATE** (ON DUPLICATE KEY UPDATE)
- Wordt gebruikt om aanwezigheid te zetten of te wijzigen
- Geeft niets terug (void)

Kortom een repository:

- vormt de **brug tussen applicatie en database**
- verbergt SQL achter duidelijke methodes
- zorgt voor schonere, beter gestructureerde code

app/Repository/AttendanceRepository.php

app > Repositories > AttendanceRepository.php

```php
<?php
class AttendanceRepository {
  public function __construct(private PDO $pdo) {}

  public function get($meetingId, $userId): ?array {
    $sql = "SELECT um.*, s.name AS status_name
            FROM users_meetings um
            JOIN status s ON s.id = um.statusid
            WHERE um.meetingsid=:mid AND um.userid=:uid LIMIT 1";
    $st = $this->pdo->prepare($sql);
    $st->execute(['mid' => (int)$meetingId, 'uid' => (int)$userId]);
    $row = $st->fetch();
    return $row ?: null;
  }

  public function set($meetingId, $userId, $statusId): void {
    $sql = "INSERT INTO users_meetings (userid, meetingsid, statusid, updated_at)
            VALUES (:uid,:mid,:sid,:upd)
            ON DUPLICATE KEY UPDATE statusid=VALUES(statusid), updated_at=VALUES(updated_at)";
    $st = $this->pdo->prepare($sql);
    $st->execute([
      'uid' => (int)$userId,
      'mid' => (int)$meetingId,
      'sid' => (int)$statusId,
      'upd' => date('Y-m-d')
    ]);
  }
}
```

get($meetingId, $userId): ?array

Haalt de attendance-status op van één gebruiker voor één meeting.

Wat gebeurt er:

- Selecteert gegevens uit de koppel-/junction-tabel users_meetings
- Koppelt de tabel status om de **leesbare statusnaam** op te halen
- Filtert op:
    - meetingsid
    - userid
- Beperkt tot één resultaat (LIMIT 1)
- Geeft:
    - een array met attendance-gegevens (incl. status_name)
    - of null als de gebruiker nog geen status heeft gezet

set($meetingId, $userId, $statusId): void

Slaat de attendance-status van een gebruiker op of werkt deze bij.

Wat gebeurt er:

- Probeert eerst een **INSERT** in users_meetings
- Bestaat de combinatie (userid, meetingsid) al:
    - dan wordt automatisch een **UPDATE** uitgevoerd
- statusid en updated_at worden bijgewerkt

Dit heet een **upsert** (insert-or-update).

app/Repository/GroupRepository.php

app > Repositories > GroupRepository.php

```php
<?php
class GroupRepository {
  public function __construct(private PDO $pdo) {}

  public function listForUser($userId): array {
    $sql = "SELECT g.*
            FROM `groups` g
            JOIN users_groups ug ON ug.groupsid = g.id
            WHERE ug.userid = :uid
            ORDER BY g.id DESC";
    $st = $this->pdo->prepare($sql);
    $st->execute(['uid' => (int)$userId]);
    return $st->fetchAll();
  }

  public function create($ownerId, $name, $description, $joinCode): int {
    $sql = "INSERT INTO `groups` (userid,name,description,join_code,created_at)
            VALUES (:uid,:name,:descr,:code,:created)";
    $st = $this->pdo->prepare($sql);
    $st->execute([
      'uid' => (int)$ownerId,
      'name' => $name,
      'descr' => $description,
      'code' => $joinCode,
      'created' => date('Y-m-d')
    ]);
    $groupId = (int)$this->pdo->lastInsertId();

    $this->addMember($ownerId, $groupId);
    return $groupId;
  }

  public function addMember($userId, $groupId): void {
    $sql = "INSERT IGNORE INTO users_groups (userid, groupsid, joined_at)
            VALUES (:uid,:gid,:joined)";
    $st = $this->pdo->prepare($sql);
    $st->execute([
      'uid' => (int)$userId,
      'gid' => (int)$groupId,
      'joined' => date('Y-m-d')
    ]);
  }

  public function joinByCode($userId, $code): ?int {
    $st = $this->pdo->prepare("SELECT id FROM `groups` WHERE join_code=:code LIMIT 1");
    $st->execute(['code' => $code]);
    $row = $st->fetch();
    if (!$row) return null;

    $gid = (int)$row['id'];
    $this->addMember($userId, $gid);
    return $gid;
  }

  public function find($groupId): ?array {
    $st = $this->pdo->prepare("SELECT * FROM `groups` WHERE id=:id");
    $st->execute(['id' => (int)$groupId]);
    $row = $st->fetch();
    return $row ?: null;
  }
}
```

```php

    public function members($groupId): array {
      $sql = "SELECT u.id, u.name, u.email, ug.joined_at
              FROM users u
              JOIN users_groups ug ON ug.userid=u.id
              WHERE ug.groupsid=:gid
              ORDER BY u.name";
      $st = $this->pdo->prepare($sql);
      $st->execute(['gid' => (int)$groupId]);
      return $st->fetchAll();
    }

    public function isMember($userId, $groupId): bool {
      $st = $this->pdo->prepare("SELECT 1 FROM users_groups WHERE userid=:uid AND groupsid=:gid LIMIT 1");
      $st->execute(['uid' => (int)$userId, 'gid' => (int)$groupId]);
      return (bool)$st->fetchColumn();
    }
}
```

listForUser($userId): array

Haalt alle groepen op waar een gebruiker lid van is.

Wat gebeurt er:

- Selecteert g.* uit groups
- Join met users_groups om lidmaatschap te bepalen
- Filtert op ug.userid = :uid
- Sorteert nieuwste groep eerst (ORDER BY g.id DESC)
- Geeft een array met groepen terug

create($ownerId, $name, $description, $joinCode): int

Maakt een nieuwe groep aan.

Wat gebeurt er:

- INSERT in groups met:
    - eigenaar (userid)
    - naam, beschrijving
    - join code (om later te kunnen joinen)
    - created_at (datum van vandaag)
- Haalt het nieuwe groupId op via lastInsertId()
- Voegt daarna automatisch de eigenaar toe als lid via addMember()
- Geeft het groupId terug

Belangrijk: eigenaar = maker, maar lidmaatschap staat apart in users_groups.

addMember($userId, $groupId): void

Voegt een gebruiker toe aan een groep.

Wat gebeurt er:

- INSERT in users_groups met joined_at
- INSERT IGNORE zorgt ervoor dat:
    - als de user al lid is, er geen fout komt
    - dubbele rijen worden voorkomen

joinByCode($userId, $code): ?int

Laat een gebruiker joinen via een join code.

Wat gebeurt er:

1. Zoek de groep op met join_code
2. Bestaat die code niet → return null
3. Bestaat de groep wel:
    - haal id op
    - voeg de gebruiker toe via addMember()
    - return het groupId

find($groupId): ?array

Haalt één groep op via ID.

- SELECT * uit groups
- Geeft groep-array terug of null als niet gevonden

members($groupId): array

Haalt alle leden van een groep op.

Wat gebeurt er:

- Selecteert gebruikersgegevens (id, name, email)
- Join met users_groups om joined_at mee te nemen
- Filtert op groupId
- Sorteert alfabetisch op naam
- Geeft array met leden terug

isMember($userId, $groupId): bool

Controleert of een gebruiker lid is van een groep.

Wat gebeurt er:

- SELECT 1 met LIMIT 1 (snelle check)
- fetchColumn() → geeft iets terug of false
- Wordt omgezet naar boolean

app/Repository/MeetingRepository.php

app > Repositories > MeetingRepository.php

```php
<?php
class MeetingRepository {
  public function __construct(private PDO $pdo) {}

  public function listForGroup($groupId): array {
    $st = $this->pdo->prepare("SELECT * FROM meetings WHERE groupsid=:gid ORDER BY meeting_datetime ASC");
    $st->execute(['gid' => (int)$groupId]);
    return $st->fetchAll();
  }

  public function create($groupId, $userId, $subject, $meetingDate, $locationOrLink): int {
    $sql = "INSERT INTO meetings (groupsid, userid, subject, meeting_datetime, location_or_link, created_at)
            VALUES (:gid,:uid,:subj,:dt,:loc,:created)";
    $st = $this->pdo->prepare($sql);
    $st->execute([
      'gid' => (int)$groupId,
      'uid' => (int)$userId,
      'subj' => $subject,
      'dt' => $meetingDate,      // DATE
      'loc' => $locationOrLink ?: null,
      'created' => date('Y-m-d')
    ]);
    return (int)$this->pdo->lastInsertId();
  }

  public function nextForUser($userId): ?array {
    $sql = "SELECT m.*
            FROM meetings m
            JOIN users_groups ug ON ug.groupsid = m.groupsid
            WHERE ug.userid=:uid AND m.meeting_datetime >= :today
            ORDER BY m.meeting_datetime ASC
            LIMIT 1";
    $st = $this->pdo->prepare($sql);
    $st->execute(['uid' => (int)$userId, 'today' => date('Y-m-d')]);
    $row = $st->fetch();
    return $row ?: null;
  }

public function findById($meetingId): ?array {
    $st = $this->pdo->prepare("SELECT * FROM meetings WHERE id=:id LIMIT 1");
    $st->execute(['id' => (int)$meetingId]);
    $row = $st->fetch();
    return $row ?: null;
  }
}
```

listForGroup($groupId): array

Haalt alle meetings op voor één groep.

Wat gebeurt er:

- Selecteert meetings uit meetings
- Filtert op groupsid
- Sorteert chronologisch op datum en tijd (ASC)
- Geeft een array met meetings terug

create(...): int

Maakt een nieuwe meeting aan.

Wat gebeurt er:

- INSERT in meetings
- Koppelt de meeting aan:
    - een groep (groupsid)
    - de gebruiker die de meeting aanmaakt (userid)
- Slaat onderwerp, datum/tijd en locatie of link op
- location_or_link wordt null als er niets is ingevuld
- created_at wordt automatisch gezet
- Geeft het **ID van de nieuwe meeting** terug

nextForUser($userId): ?array

Zoekt de eerstvolgende meeting voor een gebruiker.

Wat gebeurt er:

- Koppelt meetings aan users_groups
- Zoekt meetings van groepen waar de gebruiker lid van is
- Alleen meetings vanaf vandaag (>= vandaag)
- Sorteert op eerstvolgende datum
- Beperkt tot één resultaat (LIMIT 1)
- Geeft:
    - een meeting-array terug
    - of null als er geen toekomstige meeting is

findById($meetingId): ?array

Haalt één meeting op via het ID.

Wat gebeurt er:

- SELECT op meetings
- Beperkt tot één record
- Geeft:
    - een meeting-array
    - of null als de meeting niet bestaat

app/Repository/MetaRepository.php

app > Repositories > MetaRepository.php

```php
<?php
class MetaRepository {
  public function __construct(private PDO $pdo) {}

  public function priorities(): array {
    return $this->pdo->query("SELECT id, name FROM priority ORDER BY id")->fetchAll();
  }

  public function statusesForTable($tableId): array {
    $st = $this->pdo->prepare("SELECT id, name FROM status WHERE tableId = :t ORDER BY id");
    $st->execute(['t' => (int)$tableId]);
    return $st->fetchAll();
  }

  public function statusIdByName($tableId, $name): ?int {
    $st = $this->pdo->prepare("SELECT id FROM status WHERE tableId=:t AND name=:n LIMIT 1");
    $st->execute(['t' => (int)$tableId, 'n' => $name]);
    $row = $st->fetch();
    return $row ? (int)$row['id'] : null;
  }
}
```

priorities(): array

Haalt alle prioriteiten op.

Wat gebeurt er:

- Selecteert id en name uit de tabel priority
- Sorteert op id
- Geeft een array terug met alle prioriteiten

statusesForTable($tableId): array

Haalt alle statussen op die horen bij een specifieke tabel.

Wat gebeurt er:

- Selecteert statussen uit status
- Filtert op tableId
- Sorteert op id
- Geeft een array met statussen terug

Waarom tableId?

- Verschillende tabellen (bijv. tasks, meetings) kunnen hun **eigen statussen** hebben
- Zo blijft het statussysteem flexibel en herbruikbaar

statusIdByName($tableId, $name): ?int

Zoekt het ID van een status op basis van de naam.

Wat gebeurt er:

- Zoekt in status naar één record met:
    - de juiste tableId
    - de opgegeven name
- Beperkt tot één resultaat (LIMIT 1)
- Geeft:
    - het statusId terug als integer
    - of null als de status niet bestaat

app/Repository/TaskRepository.php

app > Repositories > TaskRepository.php

```php
<?php
class TaskRepository {
  public function __construct(private PDO $pdo) {}

public function listForUser($userId, $statusId = null): array {
  $sql = "SELECT
            t.*,
            p.name AS priority_name,
            s.name AS status_name
          FROM tasks t
          JOIN priority p ON p.id = t.priorityid
          JOIN status s ON s.id = t.statusid
          WHERE t.userid = :uid";

  $params = ['uid' => (int)$userId];

  if ($statusId) {
    $sql .= " AND t.statusid = :sid";
    $params['sid'] = (int)$statusId;
  }

  $sql .= " ORDER BY t.id DESC";

  $st = $this->pdo->prepare($sql);
  $st->execute($params);
  return $st->fetchAll();
}

  public function create($userId, $title, $description, $deadline, $priorityId, $statusId): int {
    $sql = "INSERT INTO tasks (userid,title,description,deadline,priorityid,statusid,created_at)
            VALUES (:uid,:title,:descr,:deadline,:pid,:sid,:created)";
    $st = $this->pdo->prepare($sql);
    $st->execute([
      'uid' => (int)$userId,
      'title' => $title,
      'descr' => $description ?: null,
      'deadline' => $deadline ?: null,
      'pid' => (int)$priorityId,
      'sid' => (int)$statusId,
      'created' => date('Y-m-d')
    ]);
    return (int)$this->pdo->lastInsertId();
  }

  public function update($id, $userId, $title, $description, $deadline, $priorityId, $statusId): void {
    $sql = "UPDATE tasks SET title=:title, description=:descr, deadline=:deadline,
            priorityid=:pid, statusid=:sid
            WHERE id=:id AND userid=:uid";
    $st = $this->pdo->prepare($sql);
    $st->execute([
      'id' => (int)$id,
      'uid' => (int)$userId,
      'title' => $title,
      'descr' => $description ?: null,
      'deadline' => $deadline ?: null,
      'pid' => (int)$priorityId,
      'sid' => (int)$statusId,
    ]);
  }

  public function delete($id, $userId): void {
    $st = $this->pdo->prepare("DELETE FROM tasks WHERE id=:id AND userid=:uid");
    $st->execute(['id' => (int)$id, 'uid' => (int)$userId]);
  }

```

```php

  public function countOpen($userId, $doneStatusId): int {
    $st = $this->pdo->prepare("SELECT COUNT(*) AS c FROM tasks WHERE userid=:uid AND statusid != :done");
    $st->execute(['uid' => (int)$userId, 'done' => (int)$doneStatusId]);
    return (int)$st->fetch()['c'];
  }

  public function donePercentage($userId, $doneStatusId): float {
    $st = $this->pdo->prepare("SELECT COUNT(*) AS c FROM tasks WHERE userid=:uid");
    $st->execute(['uid' => (int)$userId]);
    $total = (int)$st->fetch()['c'];
    if ($total === 0) return 0;

    $st2 = $this->pdo->prepare("SELECT COUNT(*) AS c FROM tasks WHERE userid=:uid AND statusid=:done");
    $st2->execute(['uid' => (int)$userId, 'done' => (int)$doneStatusId]);
    $done = (int)$st2->fetch()['c'];

    return round(($done / $total) * 100, 1);
  }
}
```

listForUser($userId, $statusId = null): array

Haalt een lijst taken op voor één gebruiker.

Wat doet deze methode:

- Selecteert taken uit tasks voor userid = :uid
- Koppelt extra tabellen om **leesbare namen** toe te voegen:
    - priority → priority_name
    - status → status_name
- Optioneel filter:
    - als $statusId is meegegeven, wordt gefilterd op die status
- Sorteert op nieuwste taak eerst: ORDER BY t.id DESC
- Geeft een **array met taken** terug (fetchAll())

Belangrijk detail: de query wordt dynamisch uitgebreid als $statusId bestaat.

create(...): int

Maakt een nieuwe taak aan.

Wat gebeurt er:

- INSERT in tasks met velden zoals title, description, deadline, priority, status
- description en deadline worden **null** als ze leeg zijn (?: null)
- created_at wordt automatisch gezet op de huidige datum
- Geeft het **ID van de nieuw aangemaakte taak** terug met lastInsertId()

update(...): void

Wijzigt een bestaande taak.

Wat gebeurt er:

- UPDATE van titel, beschrijving, deadline, priority en status
- Beveiliging: update alleen als:
    - id = :id én userid = :uid

Dit voorkomt dat een gebruiker taken van iemand anders kan aanpassen.

delete($id, $userId): void

Verwijdert een taak.

Wat gebeurt er:

- DELETE van tasks
- Alleen als id én userid matchen
  Ook dit is een beveiliging tegen het verwijderen van andermans taken.

countOpen($userId, $doneStatusId): int

Telt hoeveel taken nog “open” zijn.

Wat gebeurt er:

- COUNT(*) voor taken van de gebruiker
- Waar status **niet** gelijk is aan de “done”-status (statusid != :done)
- Geeft een integer terug

Let op: “open” betekent hier alles behalve done (dus ook “in progress” etc.)

donePercentage($userId, $doneStatusId): float

Berekent het percentage afgeronde taken.

Stappen:

1. Tel totaal aantal taken voor de gebruiker
    - Als totaal 0 is → return 0 (voorkomt delen door 0)
2. Tel aantal taken met status “done”
3. Bereken: (done / total) * 100
4. Rond af op 1 decimaal

Resultaat: een percentage zoals 66.7

app/Repository/UserRepository.php

app > Repositories > UserRepository.php

```php
<?php
class UserRepository {
  public function __construct(private PDO $pdo) {}

  public function findByEmail($email): ?User {
    $sql = "SELECT u.*, r.name AS rolename
            FROM users u
            JOIN role r ON r.id = u.roleid
            WHERE u.email = :email LIMIT 1";
    $st = $this->pdo->prepare($sql);
    $st->execute(['email' => $email]);
    $row = $st->fetch();
    if (!$row) return null;

    return new User($row['id'], $row['name'], $row['email'], $row['password_hash'], $row['rolename'], $row['created_at']);
  }

  public function create($name, $email, $passwordHash, $roleId = 1): int {
    $sql = "INSERT INTO users (name, email, password_hash, roleid, created_at)
            VALUES (:name,:email,:ph,:roleid,:created)";
    $st = $this->pdo->prepare($sql);
    $st->execute([
      'name' => $name,
      'email' => $email,
      'ph' => $passwordHash,
      'roleid' => $roleId,
      'created' => date('Y-m-d')
    ]);
    return (int)$this->pdo->lastInsertId();
  }

  public function findById($id): ?User {
    $sql = "SELECT u.*, r.name AS rolename
            FROM users u
            JOIN role r ON r.id = u.roleid
            WHERE u.id = :id LIMIT 1";
    $st = $this->pdo->prepare($sql);
    $st->execute(['id' => $id]);
    $row = $st->fetch();
    if (!$row) return null;

    return new User($row['id'], $row['name'], $row['email'], $row['password_hash'], $row['rolename'], $row['created_at']);
  }
}
```

findByEmail($email)

Zoekt één gebruiker op basis van het e-mailadres.

Wat gebeurt er:

- Er wordt gezocht in de users tabel
- De role tabel wordt gekoppeld om de rolnaam op te halen
- Er wordt maximaal één record opgehaald
- Bestaat de gebruiker niet → null
- Bestaat de gebruiker wel → een **User-object**

create($name, $email, $passwordHash, $roleId = 1)

Maakt een nieuwe gebruiker aan in de database.

Wat gebeurt er:

- Gebruikersgegevens worden opgeslagen in users
- Het wachtwoord wordt als **hash** opgeslagen
- De rol krijgt standaard waarde 1
- De aanmaakdatum wordt automatisch gezet
- De methode geeft het **ID van de nieuwe gebruiker** terug

findById($id)

Zoekt één gebruiker op basis van het ID.

Wat gebeurt er:

- Er wordt gezocht in users
- De bijbehorende rol wordt opgehaald
- Geen resultaat → null
- Wel resultaat → een **User-object**

App/Services/

Services bevatten de **businesslogica** van de applicatie.
Ze zitten **tussen controllers en repositories** in.

- **Controllers** → verwerken requests (input/output)
- **Services** → beslissen *wat* er moet gebeuren
- **Repositories** → voeren database-acties uit

Services combineren repositories, doen validaties en regelen de applicatielogica.

Waarom services gebruiken?

- Controllers blijven **dun en overzichtelijk**
- Repositories blijven **database-gericht**
- Logica is **herbruikbaar**
- Betere scheiding van verantwoordelijkheden

App/Services/AuthService.php

app > Services > AuthService.php

```php
<?php
class AuthService {
  private $userRepository;

  public function __construct(UserRepository $userRepository) {
    $this->userRepository = $userRepository;
  }

  public function register($name, $email, $password) {
    $name = trim($name);
    $email = trim($email);

    if ($name === '' || $email === '' || $password === '') {
      throw new Exception("Vul alle velden in.");
    }
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      throw new Exception("Ongeldig e-mailadres.");
    }
    if ($this->userRepository->findByEmail($email)) {
      throw new Exception("E-mail bestaat al.");
    }

    $hash = password_hash($password, PASSWORD_DEFAULT);
    $id = $this->userRepository->create($name, $email, $hash, 1);
    $user = $this->userRepository->findById($id);

    $_SESSION['user'] = [
      'id' => $user->getId(),
      'name' => $user->getName(),
      'email' => $user->getEmail(),
      'role' => $user->getRole(),
    ];

    return $user;
  }

  public function login($email, $password) {
    $email = trim($email);
    if ($email === '' || $password === '') {
      throw new Exception("Vul e-mail en wachtwoord in.");
    }

    $user = $this->userRepository->findByEmail($email);
    if (!$user || !$user->verifyPassword($password)) {
      throw new Exception("Onjuiste gegevens.");
    }

    $_SESSION['user'] = [
      'id' => $user->getId(),
      'name' => $user->getName(),
      'email' => $user->getEmail(),
      'role' => $user->getRole(),
    ];

    return $user;
  }

  public function logout() {
    session_destroy();
  }
}
```

register($name, $email, $password)

Registreert een nieuwe gebruiker.

Wat gebeurt er stap voor stap:

1. Invoer opschonen (trim)
2. Controleren of alle velden zijn ingevuld
3. Controleren of het e-mailadres geldig is
4. Controleren of het e-mailadres nog niet bestaat
5. Wachtwoord veilig **hashen**
6. Nieuwe gebruiker aanmaken via UserRepository
7. Gebruiker opnieuw ophalen als User-object
8. Gebruikersgegevens opslaan in de **session**
9. Het User-object teruggeven

Fouten worden afgehandeld met Exceptions.

login($email, $password)

Logt een bestaande gebruiker in.

Wat gebeurt er:

1. Invoer controleren
2. Gebruiker ophalen via findByEmail
3. Controleren of het wachtwoord klopt
4. Gebruiker opslaan in de **session**
5. Het User-object teruggeven

Bij onjuiste gegevens wordt een Exception gegooid.

logout()

Beëindigt de sessie en logt de gebruiker uit.

App/Services/DashboardService.php

app > Services > DashboardService.php

```php
<?php
class DashboardService {
    public function __construct(
        private TaskRepository $taskRepo,
        private MeetingRepository $meetingRepo,
        private MetaRepository $metaRepo
    ) {}

    public function getOpenTaskCount($userId) {
        $doneId = $this->metaRepo->statusIdByName(1, 'done');
        if (!$doneId) return 0;
        return $this->taskRepo->countOpen($userId, $doneId);
    }

    public function getDoneTaskPercentage($userId) {
        $doneId = $this->metaRepo->statusIdByName(1, 'done');
        if (!$doneId) return 0;
        return $this->taskRepo->donePercentage($userId, $doneId);
    }

    public function getNextMeeting($userId) {
        return $this->meetingRepo->nextForUser($userId);
    }
}
```

getOpenTaskCount($userId)

Geeft het aantal **open taken** terug voor een gebruiker.

Wat gebeurt er:

1. De service vraagt via MetaRepository het ID op van de status "done"
2. Bestaat deze status niet → resultaat is 0
3. Via TaskRepository wordt het aantal taken geteld dat **niet** de status "done" heeft
4. Het aantal open taken wordt teruggegeven

getDoneTaskPercentage($userId)

Geeft het **percentage afgeronde taken** terug.

Wat gebeurt er:

1. Het status-ID van "done" wordt opgehaald
2. Bestaat deze status niet → resultaat is 0
3. Via TaskRepository wordt berekend welk deel van de taken afgerond is
4. Het percentage wordt teruggegeven

getNextMeeting($userId)

Haalt de **eerstvolgende meeting** op voor een gebruiker.

Wat gebeurt er:

- De service roept nextForUser() aan op MeetingRepository
- De eerstvolgende toekomstige meeting wordt teruggegeven
- Als er geen meeting is, wordt null teruggegeven

App/Public

Binnen de public map bevinden zich de php bestanden waarin je de html plaatst, om je content te gaan tonen op het scherm.

Wanneer je naar deze pagina's wil navigeren, dien je naar de volgende link te gaan:

http://localhost/studyBuddy/public/

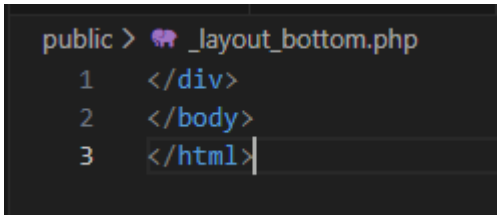
Zolang de volgende code nog niet is uitgevoerd, zul je naar de index.php gaan.

Wanneer deze wel allemaal is ingevoerd, zul je bij de bovenstaande link naar de login.php navigeren. Dit komt door onze eerdere redirect in de app/loader.php

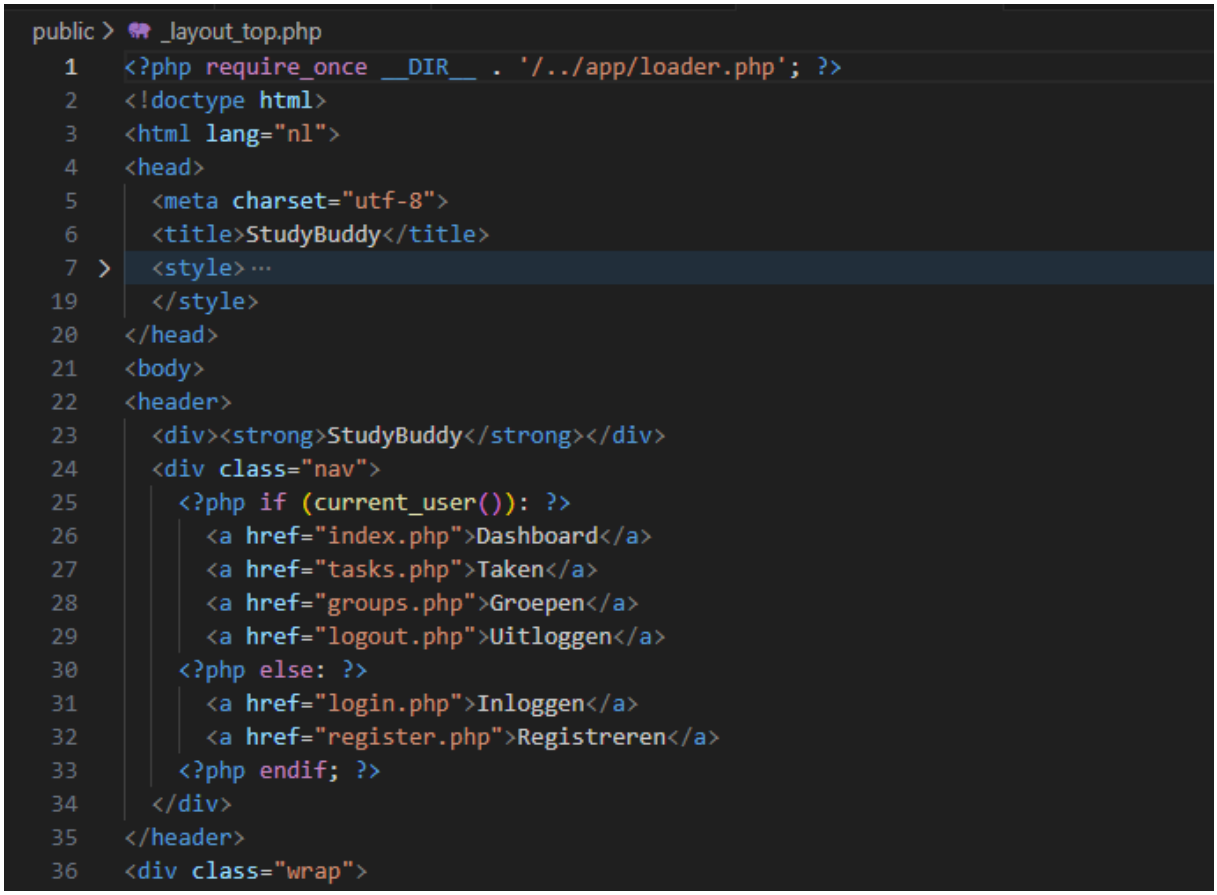
De bestanden binnen de public map, zijn de pagina's waarin je binnen jouw applicatie navigeert naar je bestanden met gekoppelde functies.

Hieronder vindt je de code voor alle pagina's, voer deze in, zodat zichtbare framework begint te ontstaan.

App/Public/_layout_bottom.php



App/Public/_layout_top.php



Plak onderstaande code tussen de <style></style> tags.

```
body{font-family:Arial;margin:0;background:#f5f6fa;}
header{background:#111827;color:#fff;padding:12px 16px;display:flex;justify-content:space-between}
.wrap{max-width:1000px;margin:16px auto;padding:0 12px;}
.card{background:#fff;padding:14px;border-radius:10px;margin:12px 0;box-shadow:0 2px 10px rgba(0,0,0,.06)}
input,select,textarea{width:100%;padding:10px;margin:6px 0;border:1px solid #ddd;border-radius:8px;box-sizing:border-box;}
button{padding:10px 12px;border:0;border-radius:8px;background:#2563eb;color:#fff;cursor:pointer;}
a{color:#2563eb;text-decoration:none}
.nav a{color:#fff;margin-right:10px;}
table{width:100%;border-collapse:collapse}
th,td{padding:8px;border-bottom:1px solid #eee;text-align:left}
.error{background:#fee2e2;color:#991b1b;padding:10px;border-radius:8px;}
```

App/Public/group.php

public > group.php

```php
<?php
require_once __DIR__ . '/../app/loader.php';
require_login();

$pdo = Database::pdo();
$groupRepo = new GroupRepository($pdo);
$meetingRepo = new MeetingRepository($pdo);

$user = current_user();
$error = null;

$groupId = isset($_GET['id']) ? (int)$_GET['id'] : 0;
if ($groupId <= 0) redirect('groups.php');

if (!$groupRepo->isMember($user['id'], $groupId)) {
  die("Geen toegang: je bent geen lid van deze groep.");
}

$group = $groupRepo->find($groupId);
if (!$group) redirect('groups.php');

// Add meeting
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  try {
    $subject = trim($_POST['subject'] ?? '');
    $date = $_POST['meeting_datetime'] ?? '';
    $loc = trim($_POST['location_or_link'] ?? '');

    if ($subject === '') throw new Exception("Onderwerp is verplicht.");
    if ($date === '') throw new Exception("Datum is verplicht.");

    $meetingRepo->create($groupId, $user['id'], $subject, $date, $loc);
    redirect("group.php?id=" . $groupId);
  } catch (Exception $e) {
    $error = $e->getMessage();
  }
}

$members = $groupRepo->members($groupId);
$meetings = $meetingRepo->listForGroup($groupId);

include '_layout_top.php';
?>
<div class="card">
  <h2>Groep: <?= e($group['name']) ?></h2>
  <p><?= e($group['description']) ?></p>
  <p><strong>Join code:</strong> <?= e($group['join_code']) ?></p>
</div>

```

```php
<div class="card">
  <h3>Leden</h3>
  <table>
    <thead><tr><th>Naam</th><th>E-mail</th><th>Joined</th></tr></thead>
    <tbody>
      <?php foreach ($members as $m): ?>
        <tr>
          <td><?= e($m['name']) ?></td>
          <td><?= e($m['email']) ?></td>
          <td><?= e($m['joined_at']) ?></td>
        </tr>
      <?php endforeach; ?>
    </tbody>
  </table>
</div>

<div class="card">
  <h3>Afspraak toevoegen</h3>
  <?php if ($error): ?><div class="error"><?= e($error) ?></div><?php endif; ?>
  <form method="post">
    <label>Onderwerp</label>
    <input name="subject" required>
    <label>Datum</label>
    <input type="date" name="meeting_datetime" required>
    <label>Locatie / link</label>
    <input name="location_or_link">
    <button>Opslaan</button>
  </form>
</div>

<div class="card">
  <h3>Afspraken</h3>
  <table>
    <thead><tr><th>Datum</th><th>Onderwerp</th><th>Locatie</th><th></th></tr></thead>
    <tbody>
      <?php foreach ($meetings as $me): ?>
        <tr>
          <td><?= e($me['meeting_datetime']) ?></td>
          <td><?= e($me['subject']) ?></td>
          <td><?= e($me['location_or_link'] ?? '-') ?></td>
          <td><a href="meeting.php?id=<?= (int)$me['id'] ?>">Open</a></td>
        </tr>
      <?php endforeach; ?>
    </tbody>
  </table>
</div>

<?php include '_layout_bottom.php'; ?>

```

App/Public/groups.php

public > groups.php

```php
<?php
require_once __DIR__ . '/../app/loader.php';
require_login();

$pdo = Database::pdo();
$groupRepo = new GroupRepository($pdo);

$user = current_user();
$error = null;

// Create group
if ($_SERVER['REQUEST_METHOD'] === 'POST' && ($_POST['action'] ?? '') === 'create') {
  try {
    $name = trim($_POST['name'] ?? '');
    $description = trim($_POST['description'] ?? '');
    if ($name === '') throw new Exception("Naam is verplicht.");
    if ($description === '') throw new Exception("Omschrijving is verplicht.");

    $joinCode = substr(bin2hex(random_bytes(8)), 0, 10);
    $groupRepo->create($user['id'], $name, $description, $joinCode);
    redirect('groups.php');
  } catch (Exception $e) {
    $error = $e->getMessage();
  }
}

// Join group by code
if ($_SERVER['REQUEST_METHOD'] === 'POST' && ($_POST['action'] ?? '') === 'join') {
  try {
    $code = trim($_POST['join_code'] ?? '');
    if ($code === '') throw new Exception("Vul een code in.");

    $gid = $groupRepo->joinByCode($user['id'], $code);
    if (!$gid) throw new Exception("Code niet gevonden.");
    redirect("group.php?id=" . (int)$gid);
  } catch (Exception $e) {
    $error = $e->getMessage();
  }
}

$groups = $groupRepo->listForUser($user['id']);

include '_layout_top.php';
?>
<div class="card">
  <h2>Groepen</h2>
  <?php if ($error): ?><div class="error"><?= e($error) ?></div><?php endif; ?>
</div>

<div class="card">
  <h3>Nieuwe groep maken</h3>
  <form method="post">
    <input type="hidden" name="action" value="create">
    <label>Groepsnaam</label>
    <input name="name" required>
    <label>Omschrijving</label>
    <input name="description" required>
    <button>Groep maken</button>
  </form>
</div>

```

```
62  <div class="card">
63    <h3>Deelnemen met code</h3>
64    <form method="post">
65      <input type="hidden" name="action" value="join">
66      <label>Join code</label>
67      <input name="join_code" required>
68      <button class="secondary">Deelnemen</button>
69    </form>
70  </div>
71
72  <div class="card">
73    <h3>Mijn groepen</h3>
74    <table>
75      <thead>
76        <tr><th>Naam</th><th>Omschrijving</th><th></th></tr>
77      </thead>
78      <tbody>
79        <?php foreach ($groups as $g): ?>
80          <tr>
81            <td><?= e($g['name']) ?></td>
82            <td><?= e($g['description']) ?></td>
83            <td><a href="group.php?id=<?= (int)$g['id'] ?>">Open</a></td>
84          </tr>
85        <?php endforeach; ?>
86      </tbody>
87    </table>
88  </div>
89
90  <?php include '_layout_bottom.php'; ?>
91
```

App/Public/index.php

public > index.php

```php
<?php
require_once __DIR__ . '/../app/loader.php';
require_login();

$pdo = Database::pdo();
$dash = new DashboardService(
  new TaskRepository($pdo),
  new MeetingRepository($pdo),
  new MetaRepository($pdo)
);

$user = current_user();
$open = $dash->getOpenTaskCount($user['id']);
$pct = $dash->getDoneTaskPercentage($user['id']);
$next = $dash->getNextMeeting($user['id']);

include '_layout_top.php';
?>
<div class="card">
  <h2>Welkom, <?= e($user['name']) ?></h2>
  <p>Openstaande taken: <strong><?= (int)$open ?></strong></p>
  <p>Afgeronde taken: <strong><?= e((string)$pct) ?>%</strong></p>
  <p>Volgende afspraak:
    <strong>
      <?php if ($next): ?>
        <?= e($next['meeting_datetime']) ?> — <?= e($next['subject']) ?>
      <?php else: ?>
        Geen afspraak gepland
      <?php endif; ?>
    </strong>
  </p>
</div>
<?php include '_layout_bottom.php'; ?>
```

App/Public/login.php

public > login.php

```php
<?php
require_once __DIR__ . '/../app/loader.php';

$pdo = Database::pdo();
$auth = new AuthService(new UserRepository($pdo));
$error = null;

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    try {
        $auth->login($_POST['email'] ?? '', $_POST['password'] ?? '');
        redirect('index.php');
    } catch (Exception $e) {
        $error = $e->getMessage();
    }
}

include '_layout_top.php';
?>
<div class="card">
    <h2>Inloggen</h2>
    <?php if ($error): ?><div class="error"><?= e($error) ?></div><?php endif; ?>
    <form method="post">
        <label>E-mail</label>
        <input name="email" type="email" required>
        <label>Wachtwoord</label>
        <input name="password" type="password" required>
        <button>Inloggen</button>
    </form>
</div>
<?php include '_layout_bottom.php'; ?>
```

App/Public/logout.php

public > logout.php

```php
<?php
require_once __DIR__ . '/../app/loader.php';
$pdo = Database::pdo();
$auth = new AuthService(new UserRepository($pdo));
$auth->logout();
redirect('login.php');

```

App/Public/meeting.php

public > meeting.php

```php
<?php
require_once __DIR__ . '/../app/loader.php';
require_login();

$pdo = Database::pdo();
$attendanceRepo = new AttendanceRepository($pdo);
$meetingRepo = new MeetingRepository($pdo);
$groupRepo = new GroupRepository($pdo);
$metaRepo = new MetaRepository($pdo);

$user = current_user();

$meetingId = isset($_GET['id']) ? (int)$_GET['id'] : 0;
if ($meetingId <= 0) redirect('groups.php');

// Meeting ophalen (via groep lijst query)
$meetingRow = null;

$meetingRow = $meetingRepo->findById($meetingId);
if (!$meetingRow) redirect('groups.php');

$groupId = (int)$meetingRow['groupsid'];
if (!$groupRepo->isMember($user['id'], $groupId)) {
  die("Geen toegang: je bent geen lid van deze groep.");
}

$statusOptions = $metaRepo->statusesForTable(2);
$error = null;

// Update attendance
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  try {
    $statusId = (int)($_POST['statusid'] ?? 0);
    if ($statusId <= 0) throw new Exception("Kies een status.");
    $attendanceRepo->set($meetingId, $user['id'], $statusId);
    redirect("meeting.php?id=" . $meetingId);
  } catch (Exception $e) {
    $error = $e->getMessage();
  }
}

$myAttendance = $attendanceRepo->get($meetingId, $user['id']);

include '_layout_top.php';
?>
<div class="card">
  <h2>Afspraak</h2>
  <p><strong>Datum:</strong> <?= e($meetingRow['meeting_datetime']) ?></p>
  <p><strong>Onderwerp:</strong> <?= e($meetingRow['subject']) ?></p>
  <p><strong>Locatie/link:</strong> <?= e($meetingRow['location_or_link'] ?? '-') ?></p>
  <p><a href="group.php?id=<?= (int)$groupId ?>">← Terug naar groep</a></p>
</div>

<div class="card">
  <h3>Mijn aanwezigheid</h3>
  <?php if ($error): ?><div class="error"><?= e($error) ?></div><?php endif; ?>

```

```
58      <form method="post">
59        <label>Status</label>
60        <select name="statusid" required>
61          <option value="">Kies...</option>
62          <?php foreach ($statusOptions as $s): ?>
63            <option value="<?= (int)$s['id'] ?>"
64              <?= $myAttendance && (int)$myAttendance['statusid'] === (int)$s['id'] ? 'selected' : '' ?>>
65              <?= e($s['name']) ?>
66            </option>
67          <?php endforeach; ?>
68        </select>
69        <button>Opslaan</button>
70      </form>
71
72      <p>
73        Huidige status:
74        <strong><?= e($myAttendance['status_name'] ?? 'nog niet ingesteld') ?></strong>
75      </p>
76    </div>
77
78    <?php include '_layout_bottom.php'; ?>
79
```

App/Public/register.php

public > register.php

```php
<?php
require_once __DIR__ . '/../app/loader.php';

$pdo = Database::pdo();
$auth = new AuthService(new UserRepository($pdo));
$error = null;

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    try {
        $auth->register($_POST['name'] ?? '', $_POST['email'] ?? '', $_POST['password'] ?? '');
        redirect('index.php');
    } catch (Exception $e) {
        $error = $e->getMessage();
    }
}

include '_layout_top.php';
?>
<div class="card">
    <h2>Registreren</h2>
    <?php if ($error): ?><div class="error"><?= e($error) ?></div><?php endif; ?>
    <form method="post">
        <label>Naam</label>
        <input name="name" required>
        <label>E-mail</label>
        <input name="email" type="email" required>
        <label>Wachtwoord</label>
        <input name="password" type="password" required>
        <button>Account maken</button>
    </form>
</div>
<?php include '_layout_bottom.php'; ?>

```

App/Public/tasks.php

public > tasks.php

```php
<?php
require_once __DIR__ . '/../app/loader.php';
require_login();

$pdo = Database::pdo();
$taskRepo = new TaskRepository($pdo);
$metaRepo = new MetaRepository($pdo);

$user = current_user();

$statusOptions = $metaRepo->statusesForTable(1); // tasks
$priorityOptions = $metaRepo->priorities();

$error = null;
$editTask = null;

// Delete
if (isset($_GET['delete'])) {
  $taskRepo->delete((int)$_GET['delete'], $user['id']);
  redirect('tasks.php');
}

// Edit load
if (isset($_GET['edit'])) {
  $id = (int)$_GET['edit'];
  $rows = $taskRepo->listForUser($user['id'], null);
  foreach ($rows as $r) {
    if ((int)$r['id'] === $id) {
      $editTask = $r;
      break;
    }
  }
}

// Create / Update
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  try {
    $title = trim($_POST['title'] ?? '');
    $description = trim($_POST['description'] ?? '');
    $deadline = $_POST['deadline'] ?? '';
    $priorityId = (int)($_POST['priorityid'] ?? 0);
    $statusId = (int)($_POST['statusid'] ?? 0);

    if ($title === '') throw new Exception("Titel is verplicht.");
    if ($priorityId <= 0) throw new Exception("Kies een prioriteit.");
    if ($statusId <= 0) throw new Exception("Kies een status.");

    if ($deadline === '') $deadline = null;

    $id = $_POST['id'] ?? '';
    if ($id !== '') {
      $taskRepo->update((int)$id, $user['id'], $title, $description, $deadline, $priorityId, $statusId);
    } else {
      $taskRepo->create($user['id'], $title, $description, $deadline, $priorityId, $statusId);
    }

    redirect('tasks.php');
  } catch (Exception $e) {
    $error = $e->getMessage();
  }
}

```

public > tasks.php

```php
}

// Filter
$filterStatus = isset($_GET['status']) && $_GET['status'] !== '' ? (int)$_GET['status'] : null;
$tasks = $taskRepo->listForUser($user['id'], $filterStatus);

include '_layout_top.php';
?>
<div class="card">
  <h2>Mijn taken</h2>

  <?php if ($error): ?><div class="error"><?= e($error) ?></div><?php endif; ?>

  <form method="get" style="margin-bottom:12px;">
    <label>Filter op status</label>
    <select name="status">
      <option value="">(alles)</option>
      <?php foreach ($statusOptions as $s): ?>
        <option value="<?= (int)$s['id'] ?>" <?= ($filterStatus === (int)$s['id']) ? 'selected' : '' ?>>
          <?= e($s['name']) ?>
        </option>
      <?php endforeach; ?>
    </select>
    <button type="submit" class="secondary">Filter</button>
    <a href="tasks.php" style="margin-left:10px;">Reset</a>
  </form>
</div>

<div class="card">
  <h3><?= $editTask ? 'Taak bewerken' : 'Nieuwe taak' ?></h3>

  <form method="post">
    <?php if ($editTask): ?>
      <input type="hidden" name="id" value="<?= (int)$editTask['id'] ?>">
    <?php endif; ?>

    <label>Titel</label>
    <input name="title" required value="<?= e($editTask['title'] ?? '') ?>">

    <label>Beschrijving</label>
    <textarea name="description"><?= e($editTask['description'] ?? '') ?></textarea>

    <label>Deadline</label>
    <input type="date" name="deadline" value="<?= e($editTask['deadline'] ?? '') ?>">

    <label>Prioriteit</label>
    <select name="priorityid" required>
      <option value="">Kies...</option>
      <?php foreach ($priorityOptions as $p): ?>
        <option value="<?= (int)$p['id'] ?>"
          <?= isset($editTask['priorityid']) && (int)$editTask['priorityid'] === (int)$p['id'] ? 'selected' : '' ?>>
          <?= e($p['name']) ?>
        </option>
      <?php endforeach; ?>
    </select>

    <label>Status</label>
    <select name="statusid" required>
      <option value="">Kies...</option>
      <?php foreach ($statusOptions as $s): ?>
        <option value="<?= (int)$s['id'] ?>"
          <?= isset($editTask['statusid']) && (int)$editTask['statusid'] === (int)$s['id'] ? 'selected' : '' ?>>
          <?= e($s['name']) ?>
        </option>
      <?php endforeach; ?>
    </select>
```

```
      <button><?= $editTask ? 'Opslaan' : 'Toevoegen' ?></button>
      <?php if ($editTask): ?>
        <a href="tasks.php" style="margin-left:10px;">Annuleren</a>
      <?php endif; ?>
    </form>
  </div>

  <div class="card">
    <h3>Overzicht</h3>
    <table>
      <thead>
        <tr>
          <th>Titel</th>
          <th>Deadline</th>
          <th>Prioriteit</th>
          <th>Status</th>
          <th></th>
        </tr>
      </thead>
      <tbody>
      <?php foreach ($tasks as $t): ?>
        <tr>
          <td><?= e($t['title']) ?></td>
          <td><?= e($t['deadline'] ?? '-') ?></td>
          <td><?= e($t['priority_name'] ?? '-') ?></td>
          <td><?= e($t['status_name'] ?? '-') ?></td>
          <td>
            <a href="tasks.php?edit=<?= (int)$t['id'] ?>">Bewerk</a> |
            <a href="tasks.php?delete=<?= (int)$t['id'] ?>" onclick="return confirm('Taak verwijderen?')">Verwijder</a>
          </td>
        </tr>
      <?php endforeach; ?>
      </tbody>
    </table>

  </div>

  <?php include '_layout_bottom.php'; ?>

```