Structure de projet Web

Organiser ses fichiers et son code

HTML / CSS / JavaScript / PHP

Pourquoi s'organiser ?

Un projet mal organise devient rapidement **ingerable**.

Jour 1

```
index.php
style.css
script.js

 3 fichiers
 "Ca va"
```

Jour 30

```
index.php
style.css
style2.css
old_style.css
script.js
test.js
page1.php
page2.php
page2_v2.php
functions.php
img1.jpg
logo.png
logo_v2.png
 13 fichiers
 "Je m'y
  retrouve
  plus"
```

La regle d'or

Un dossier = un role. Un fichier = une responsabilite.

Ne jamais tout mettre a la racine du projet.

Objectifs :

- Retrouver un fichier en **moins de 5 secondes**
- Savoir **ou creer** un nouveau fichier
- Permettre a un coequipier de **comprendre le projet** sans explication

Structure recommandee

```
mon-projet/
├── index.php
├── assets/
│   ├── css/
│   │   └── style.css
│   ├── js/
│   │   └── main.js
│   └── img/
│       ├── logo.png
│       └── hero.jpg
├── pages/
│   ├── login.php
│   ├── register.php
│   └── dashboard.php
├── includes/
│   ├── header.php
│   ├── footer.php
│   └── nav.php
├── config/
│   └── database.php
└── sql/
    └── schema.sql
```

Explication de chaque dossier

Dossier	Contenu	Exemple
`/` (racine)	Point d'entree	`index.php`
`assets/css/`	Feuilles de style	`style.css`
`assets/js/`	Scripts JavaScript	`main.js`
`assets/img/`	Images et icones	`logo.png`
`pages/`	Pages PHP du site	`login.php`
`includes/`	Fragments reutilisables	`header.php`
`config/`	Configuration	`database.php`
`sql/`	Scripts de base de donnees	`schema.sql`

Le dossier assets/

Tout ce qui concerne l'apparence et le comportement cote client.

```
assets/
├── css/
│   ├── style.css          <-- Styles globaux
│   ├── login.css          <-- Styles specifiques
│   └── components.css     <-- Boutons, cartes, etc.
├── js/
│   ├── main.js            <-- Script principal
│   ├── validation.js      <-- Validation formulaires
│   └── search.js          <-- Logique de recherche
└── img/
    ├── logo.png
    ├── hero.jpg
    └── icons/
        ├── search.svg
        └── user.svg
```

Le dossier pages/

Une page = un fichier PHP.

```
pages/
├── login.php              <-- Formulaire de connexion
├── register.php           <-- Formulaire d'inscription
├── dashboard.php          <-- Tableau de bord
├── opportunities.php      <-- Liste des opportunites
├── opportunity-detail.php <-- Detail d'une opportunite
├── profile.php            <-- Profil utilisateur
└── search.php             <-- Page de recherche
```

> *Nommer les fichiers en fonction de ce qu'ils affichent, pas de ce qu'ils font techniquement.*

Le dossier includes/

Les morceaux de pages reutilises partout.

```
includes/
├── header.php     <-- <head>, meta, liens CSS
├── nav.php        <-- Barre de navigation
├── footer.php     <-- Pied de page, scripts JS
└── functions.php  <-- Fonctions utilitaires PHP
```

Utilisation dans une page :

```php
<?php require_once 'includes/header.php'; ?>
<?php require_once 'includes/nav.php'; ?>

<main>
    <h1>Bienvenue</h1>
    <!-- Contenu de la page -->
</main>

<?php require_once 'includes/footer.php'; ?>
```

Le dossier config/

Les parametres qui changent selon l'environnement.

```php
// config/database.php

$host = 'localhost';
$dbname = 'mobilite_internationale';
$user = 'root';
$password = '';

try {
    $pdo = new PDO(
        "mysql:host=$host;dbname=$dbname;charset=utf8mb4",
        $user,
        $password
    );
    $pdo->setAttribute(PDO::ATTR_ERRMODE,
                       PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die('Erreur de connexion : ' . $e->getMessage());
}
```

Le dossier sql/

Les scripts de creation de la base de donnees.

```sql
-- sql/schema.sql

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE opportunities (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    country VARCHAR(100) NOT NULL,
    description TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

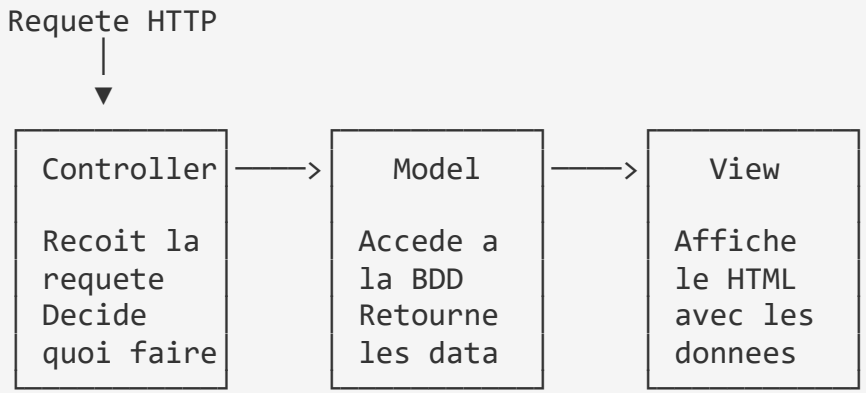
> *Garder le SQL dans un fichier permet de recreer la base facilement.*

Structure avancee : MVC

Quand le projet grandit, on separe la **logique metier** de **l'affichage**.

```
mon-projet/
├── index.php              <-- Routeur (point d'entree unique)
├── controllers/
│   ├── AuthController.php
│   └── OpportunityController.php
├── models/
│   ├── User.php
│   └── Opportunity.php
├── views/
│   ├── layout.php
│   ├── auth/
│   │   ├── login.php
│   │   └── register.php
│   └── opportunities/
│       ├── list.php
│       └── detail.php
├── assets/
├── config/
└── sql/
```

MVC : qui fait quoi ?



MVC : exemple concret

Controller - Recoit et decide :

```php
// controllers/OpportunityController.php

require_once 'models/Opportunity.php';

function listOpportunities($pdo) {
    $opportunities = getAllOpportunities($pdo);
    require 'views/opportunities/list.php';
}
```

MVC : exemple concret

Model - Accede aux donnees :

```php
// models/Opportunity.php

function getAllOpportunities($pdo) {
    $stmt = $pdo->query(
        'SELECT * FROM opportunities ORDER BY created_at DESC'
    );
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

View - Affiche le HTML :

```php
<!-- views/opportunities/list.php -->
<h1>Opportunites</h1>
<?php foreach ($opportunities as $opp): ?>
    <div class="card">
        <h2><?= htmlspecialchars($opp['title']) ?></h2>
        <p><?= htmlspecialchars($opp['country']) ?></p>
    </div>
<?php endforeach; ?>
```

Nommage des fichiers

Des regles simples pour ne jamais hesiter.

Type	Convention	Exemple
Pages PHP	kebab-case	`opportunity-detail.php`
Classes PHP	PascalCase	`UserController.php`
CSS	kebab-case	`main-style.css`
JS	camelCase ou kebab-case	`formValidation.js`
Images	kebab-case descriptif	`hero-banner.jpg`
SQL	kebab-case	`schema.sql`

Nommage : les erreurs classiques

A eviter	Pourquoi	Mieux
`page2.php`	Pas descriptif	`register.php`
`style (2).css`	Espaces + parentheses	`login.css`
`HEADER.PHP`	Majuscules	`header.php`
`final_v3_ok.php`	Versionning dans le nom	Utiliser Git
`a.js`	Incomprehensible	`validation.js`
`IMG_20250209.jpg`	Nom de camera	`campus-munich.jpg`

Organisation du HTML

Un fichier HTML bien structure

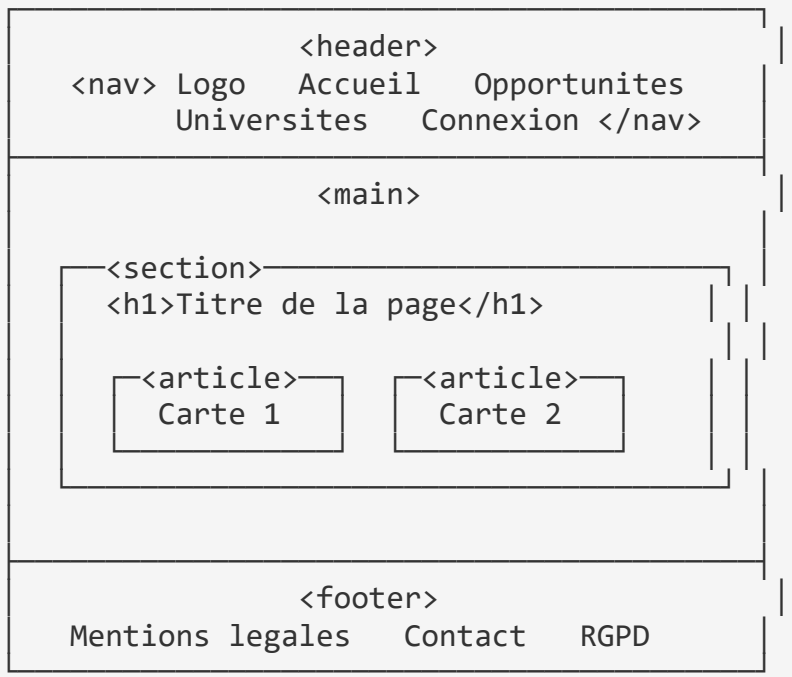
```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, initial-scale=1.0">
    <title>Mobilite Internationale</title>
    <link rel="stylesheet" href="assets/css/style.css">
</head>
<body>
    <header><!-- Navigation --></header>
    <main><!-- Contenu principal --></main>
    <footer><!-- Pied de page --></footer>

    <script src="assets/js/main.js"></script>
</body>
</html>
```

Organisation du HTML

Regles a respecter

Regle	Raison
CSS dans `<head>`	Evite le flash de contenu non style
JS avant `</body>`	La page se charge avant les scripts
Indentation coherente	Lisibilite (2 ou 4 espaces, pas les deux)
Balises semantiques	`<nav>`, `<main>`, `<article>`, `<section>`
Attributs `alt` sur `<img>`	Accessibilite

HTML : balises semantiques



Organisation du CSS

Un fichier CSS bien structure

```css
/* ================================
   1. RESET / BASE
   ================================ */
* { margin: 0; padding: 0; box-sizing: border-box; }

/* ================================
   2. TYPOGRAPHIE
   ================================ */
body { font-family: 'Segoe UI', sans-serif; }
h1 { font-size: 2rem; }

/* ================================
   3. LAYOUT (structure de page)
   ================================ */
header { }
main { }
footer { }
```

Organisation du CSS (suite)

```css
/* ================================
   4. COMPOSANTS (reutilisables)
   ================================ */
.btn { }
.card { }
.form-group { }

/* ================================
   5. PAGES SPECIFIQUES
   ================================ */
.login-form { }
.dashboard-stats { }

/* ================================
   6. RESPONSIVE
   ================================ */
@media (max-width: 768px) { }
```

CSS : bonnes pratiques

Pratique	Exemple
Noms descriptifs	`.card-opportunity` pas `.box1`
Pas d'ID pour le style	`#header` -> `.site-header`
Un composant = un bloc	`.card`, `.card-title`, `.card-body`
Eviter `!important`	Revoir la specificite a la place
Pas de style inline	`style="..."` -> fichier CSS

CSS : nommage des classes

Convention BEM (Block Element Modifier)

```css
/* Block */
.card { }

/* Element (partie du block) */
.card__title { }
.card__image { }
.card__body { }

/* Modifier (variante) */
.card--highlighted { }
.card--small { }
```

```html
<div class="card card--highlighted">
    <h2 class="card__title">TU Munich</h2>
    <img class="card__image" src="..." alt="...">
    <p class="card__body">Description...</p>
</div>
```

Organisation du JavaScript

Un fichier JS bien structure

```
// ================================
// 1. CONSTANTES ET CONFIGURATION
// ================================
const API_URL = '/api';

// ================================
// 2. FONCTIONS UTILITAIRES
// ================================
function formatDate(date) {
    return new Date(date).toLocaleDateString('fr-FR');
}

// ================================
// 3. FONCTIONS PRINCIPALES
// ================================
function loadOpportunities() {
    // ...
}

// ================================
// 4. ECOUTEURS D'EVENEMENTS
// ================================
document.addEventListener('DOMContentLoaded', () => {
    loadOpportunities();
});
```

JavaScript : bonnes pratiques

Pratique	Mauvais	Bon
Variables	`var x = 5;`	`const x = 5;` ou `let x = 5;`
Nommage	`a`, `temp`, `data2`	`userEmail`, `searchResults`
Fonctions	`function f()`	`function validateEmail()`
Comparaison	`==`	`===`
Evenements	`onclick="..."`	`addEventListener()`

JavaScript : separation des responsabilites

Ne pas melanger JS et HTML

```
<!-- Mauvais -->
<button onclick="deleteUser(5)">Supprimer</button>

<!-- Bon -->
<button class="btn-delete" data-id="5">Supprimer</button>
```

```
// Dans un fichier JS separe
document.querySelectorAll('.btn-delete')
    .forEach(btn => {
        btn.addEventListener('click', (e) => {
            const id = e.target.dataset.id;
            deleteUser(id);
        });
    });
```

Organisation du PHP

Regles de base

Regle	Raison
`<?php` complet	Jamais `<?` (short tags)
Un `require` en haut	Dependances visibles
Pas de HTML dans la logique	Separation des roles
`htmlspecialchars()`	Toujours echapper les sorties
Requetes preparees	Eviter les injections SQL

PHP : securite de base

```php
// MAUVAIS - Injection SQL
$sql = "SELECT * FROM users
        WHERE email = '$_POST[email]'";

// BON - Requete preparee
$stmt = $pdo->prepare(
    'SELECT * FROM users WHERE email = :email'
);
$stmt->execute(['email' => $_POST['email']]);
```

```php
// MAUVAIS - Faille XSS
echo $_GET['search'];

// BON - Echappement
echo htmlspecialchars($_GET['search'], ENT_QUOTES, 'UTF-8');
```

PHP : inclure ses fichiers

Ordre des inclusions dans une page

```php
<?php
// 1. Configuration
require_once 'config/database.php';

// 2. Fonctions / Modeles
require_once 'includes/functions.php';
require_once 'models/Opportunity.php';

// 3. Logique de la page
$opportunities = getAllOpportunities($pdo);

// 4. Affichage
require_once 'includes/header.php';
require_once 'includes/nav.php';
?>

<main>
    <!-- Contenu avec les donnees -->
</main>

<?php require_once 'includes/footer.php'; ?>
```

Gestion des images

Regles pratiques

Regle	Detail
Noms descriptifs	`campus-munich.jpg` pas `IMG_001.jpg`
Taille optimisee	Pas de photo 4000x3000 pour un thumbnail
Format adapte	JPG pour photos, PNG pour logos, SVG pour icones
Dossier organise	Sous-dossiers si > 10 images

Gestion des images (suite)

```
assets/img/
├── logo.png              <-- Logo du site
├── hero-banner.jpg       <-- Image d'accueil
├── universities/
│   ├── tu-munich.jpg
│   ├── oxford.jpg
│   └── mit.jpg
├── icons/
│   ├── search.svg
│   ├── user.svg
│   └── arrow.svg
└── placeholders/
    └── no-image.png      <-- Image par defaut
```

Commentaires : quand et comment

Commenter le pourquoi, pas le quoi.

```php
// INUTILE - on voit deja ce que fait le code
// Incrementer i de 1
$i++;

// UTILE - explique une decision
// Limite a 10 resultats pour eviter
// la surcharge sur mobile
$stmt = $pdo->prepare(
    'SELECT * FROM opportunities LIMIT 10'
);
```

Commentaires : en-tete de fichier

```php
<?php
/**
 * Gestion des opportunites de mobilite.
 *
 * Fonctions CRUD pour la table opportunities.
 * Utilise PDO pour les requetes preparees.
 */

function getAllOpportunities($pdo) {
    // ...
}

function getOpportunityById($pdo, $id) {
    // ...
}
```

> *Un commentaire en haut de fichier aide a comprendre son role sans lire tout le code.*

Git : les bases de l'organisation

Versionner son code, pas ses fichiers temporaires.

Fichier `.gitignore` a la racine :

```
# Fichiers systeme
.DS_Store
Thumbs.db

# Editeur
.vscode/
.idea/

# Configuration locale
config/database.php

# Dependances
vendor/
node_modules/

# Fichiers uploades par les utilisateurs
uploads/
```

Git : messages de commit

Mauvais	Bon
`update`	`Ajouter formulaire d'inscription`
`fix`	`Corriger la validation email`
`changes`	`Ajouter pagination sur la liste`
`wip`	`Creer le modele Opportunity`
`asdfgh`	Ne pas committer n'importe quoi

> *Committer **souvent** avec des messages **clairs**.*