Concept dev web

- Nouvelle page

Nouvelle page

Notes techniques — Site Erasmus+

Ce que nous avons mis en place, pourquoi, et comment ça fonctionne — expliqué simplement.

Migration vers Fat Free Framework (F3)

1. □□ Le contexte — d’où on part, où on va

Le site Erasmus+ a démarré en **PHP pur** : chaque page était un fichier PHP séparé (`articles.php`, `contacts.php`...). Simple à démarrer, mais difficile à faire évoluer sur le long terme.

On a décidé de migrer vers une architecture plus structurée, avec un **framework léger**. En bonus, on en a profité pour avoir de vraies URLs lisibles.

□ Avant (URLs “moches”)	□ Après (URLs propres)
`erasmus.thymon.fr/article.php?id=corogna-2024`	`erasmus.thymon.fr/article/corogna-2024`

Ce changement nécessitait un **point d’entrée unique** : toutes les requêtes passent désormais par `index.php`, qui distribue le travail selon l’URL demandée. C’est ce que fait F3.

---

2. □□ Le patron MVC — Modèle · Vue · Contrôleur

MVC est une façon d'organiser son code en **3 rôles bien séparés**, pour ne pas tout mélanger dans un seul fichier.

□□ Analogie — Un restaurant

- **Modèle** = la cuisine. Il sait où sont les données et comment les lire (les fichiers JSON des articles).
- **Vue** = le menu et la salle. Elle s'occupe uniquement de l'apparence (les fichiers HTML).
- **Contrôleur** = le serveur. Il prend la commande (l'URL), va chercher les données, et les envoie à la bonne vue.

Schéma simple du flux :

```
Visiteur → frappe erasmus.thymon.fr/articles
   ↓
index.php — F3 lit l'URL et dit : "ça c'est pour ArticleController → liste()"
   ↓
ArticleController.php — va chercher les articles dans les JSON
   ↓
articles.html (Vue) — affiche les articles dans le HTML
   ↓
Navigateur reçoit la page finale
```

Avantage concret : si demain tu veux changer la mise en page de la liste des articles, tu touches seulement `articles.html`. Si tu veux changer comment les données sont filtrées, tu touches seulement `ArticleController.php`. Les deux ne se mélangent pas.

---

3. ⚡ Fat Free Framework (F3) — le chef d'orchestre

F3 est un **micro-framework PHP**. "Framework" = boîte à outils qui donne une structure. "Micro" = il reste très léger, sans des dizaines de fichiers de configuration.

Son rôle principal dans notre projet : **le routage**. Il lit l'URL demandée et appelle le bon contrôleur.

```
// Dans index.php — on dit à F3 : "si quelqu'un va sur /articles,
// appelle la méthode liste() dans ArticleController"
$f3->route('GET /articles', 'ArticleController->liste');
$f3->route('GET /article/@slug', 'ArticleController->detail');
// @slug = la partie variable de l'URL (ex: "corogna-2024")
```

F3 s'occupe aussi du **moteur de templates** : il lit les fichiers `.html` des vues, remplace les `{{ @variables }}` par les vraies valeurs, et produit le HTML final envoyé au navigateur.

□□ Pourquoi F3 plutôt que Laravel/Symfony ?
Les grands frameworks sont très puissants mais ont une courbe d'apprentissage importante et beaucoup de "magie" cachée. F3 fait exactement ce dont on a besoin, sans surplus.

---

4. □□ Composer — le gestionnaire de librairies

Composer est le **gestionnaire de paquets PHP**. C'est l'équivalent d'un store : il télécharge et installe des librairies créées par d'autres développeurs.

□□ Analogie — Liste de courses

Tu écris dans `composer.json` : "j'ai besoin de F3 et d'une librairie vidéo". Composer télécharge tout et le place dans `vendor/`.
On a installé deux librairies pour ce projet :

- **bcosca/fatfree** — le framework F3
- **php-ffmpeg/php-ffmpeg** — manipulation vidéo (miniatures, futures fonctionnalités)

Le dossier `vendor/` est exclu de Git (dans `.gitignore`) car il peut être recréé avec :

```
composer install
```

---

5. □□ Les URLs propres — comment ça fonctionne

Pour que toutes les URLs passent par `index.php`, on utilise un fichier `.htaccess` à la racine du site (Apache).

```
# .htaccess — la règle clé
RewriteEngine On
# Si ce n'est pas un vrai fichier ET pas un vrai dossier...
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# ...redirige vers index.php
RewriteRule .* index.php [L,QSA]
```

Quand quelqu'un va sur `/article/corogna-2024` :

1. Apache cherche un fichier `article/corogna-2024` → n'existe pas
2. La règle `.htaccess` envoie vers `index.php`
3. F3 lit l'URL, détecte la route `/article/@slug`
4. F3 appelle `ArticleController->detail()` avec `slug = "corogna-2024"`

⚠ **Piège important — Chemins relatifs**
Avec les URLs propres, les chemins d'images relatifs cassent : sur `/article/corogna-2024`, `uploads/photo.jpg` devient `/article/uploads/photo.jpg` → introuvable. Utiliser des chemins **absolus** : `/uploads/photo.jpg`.

---

6. □□ Les templates F3 — séparer le HTML du PHP

Dans l'ancien site, le PHP et le HTML étaient mélangés. Avec F3, le contrôleur prépare les données, puis les "envoie" à un fichier HTML (vue).

Syntaxe principale :

```
<!-- Afficher une variable (avec protection HTML automatique) -->
{{ @article.titre }}
```

```
<!-- Afficher du HTML pré-généré sans l'échapper -->
{{ @contenu_html | raw }}

<!-- Condition if/else -->
<check if="{{ @article.photo }}">
  <true><img src="{{ @article.photo }}" /></true>
  <false><div>Pas de photo</div></false>
</check>

<!-- Boucle foreach -->
<repeat group="{{ @articles }}" value="{{ @a }}">
  <h2>{{ @a.titre }}</h2>
</repeat>
```

F3 compile ces templates en PHP dans `tmp/`. Il ne recompile que si le fichier source a changé. Si une vue ne se met pas à jour, vider le cache en supprimant `tmp/*.php`.

□□ Analogie — Un formulaire à remplir

Le template HTML est comme un formulaire avec des cases vides. Le contrôleur PHP remplit les cases avec les vraies données. F3 fait la liaison entre les deux.

---

7. □□ Structure du projet — qui fait quoi

```
html/
├── index.php                ← Point d'entrée UNIQUE (toutes les URLs arrivent ici)
│   └── F3 lit l'URL et distribue au bon contrôleur
│
├── app/controllers/         ← CONTRÔLEURS (logique PHP)
│   ├── BaseController.php   ← Base commune (variables layout, render...)
│   ├── HomeController.php   ← Page d'accueil (/)
│   ├── ArticleController.php ← /articles et /article/@slug
│   ├── ContactController.php ← /contacts
│   └── StandardsController.php ← /standards-qualite
│
├── app/views/               ← VUES (HTML + syntaxe F3)
```

```
│   ├── layouts/
│   │   ├── main.html         ← Accueil (navbar complète + footer)
│   │   └── simple.html       ← Pages secondaires (nav simplifiée + retour)
│   ├── home.html
│   ├── articles.html
│   ├── article.html
│   ├── contacts.html
│   └── standards.html
│
├── includes/                 ← Helpers PHP partagés
│   ├── config.php            ← Constantes + chargement config.json
│   ├── functions.php         ← Fonctions utilitaires
│   └── db.php                ← Connexion DB (futures fonctionnalités)
│
├── data/                     ← DONNÉES (JSON, protégées par .htaccess)
│   ├── articles/             ← 12 fichiers JSON (un par mobilité)
│   ├── menu.json             ← Structure du menu
│   └── config.json           ← Config (email, mdp admin hashé)
│
├── vendor/                   ← Librairies Composer (ne pas modifier)
└── admin/                    ← Pages d'administration (migration F3 à venir)
```

---

8. □□ Les layouts — le “moule” des pages

Un **layout** est le cadre commun à plusieurs pages. Au lieu de recopier la navbar et le footer partout, on les met une seule fois dans le layout.

```
┌──────────────── simple.html (layout) ────────────────┐
│ <head> Tailwind, polices, meta tags...               │
│------------------------------------------------------│
│ NAVBAR sticky (logo + lien retour)                   │
│------------------------------------------------------│
│                                                      │
│     {{ @content | raw }}   ← la vue s'insère ici     │
│                                                      │
```

```
|-------------------------------------------------|
| FOOTER                                          |
|-------------------------------------------------|
| {{ @extra_scripts | raw }} ← JS optionnel        |
└─────────────────────────────────────────────────┘
```

On a deux layouts :

- **main.html** — pour la page d'accueil (navbar complète + menu déroulant)
- **simple.html** — pour les pages secondaires (navbar allégée + lien "retour")

---

9. □□ PHP 8.4 — pièges rencontrés et leçons apprises

On tourne sur **PHP 8.4** : version plus stricte, certaines choses qui "passaient" avant provoquent maintenant des erreurs.

Piège n°1 — Variable non définie = erreur

Si un template F3 affiche `{{ @ma_variable }}` et que cette variable n'a pas été définie, PHP 8.4 peut déclencher une erreur (en DEBUG=3 : page blanche / HTTP 500).

Solution : initialiser les variables optionnelles dans `BaseController.php` :

```
// Dans BaseController::render()
$f3->set('page_description', '');
$f3->set('extra_head',       '');
$f3->set('extra_scripts',   '');
```

Piège n°2 — Clé de tableau manquante

Accéder à `$tableau['ma_cle']` quand la clé n'existe pas → "Undefined array key".

Solution : garantir que toutes les clés existent, même vides :

```
// Bien : la clé 'conditions' existe toujours (tableau vide si pas de conditions)
'groupe' => [
    'intro'      => "...",
    'items'      => [...],
    'conditions' => [], // vide mais présent
]
```

Comment diagnostiquer les erreurs :

> 1. Dans `index.php`, mettre `$f3->set('DEBUG', 3)`
> 2. Ajouter `ini_set('error_log', __DIR__ . '/php_errors.log')`
> 3. Lire `php_errors.log` pour voir l'erreur exacte et la ligne
> 4. Une fois corrigé, remettre `DEBUG=0` et retirer les logs

---

10. □□ Lexique — les termes à retenir

Terme	Définition
Framework	Boîte à outils qui impose une structure au projet.
MVC	Organisation du code en 3 rôles séparés (Modèle · Vue · Contrôleur).
Route	Association entre une URL (ex: `/articles`) et le code à exécuter.
Template	Fichier HTML avec des "trous" (`{{ @variable }}`) remplis par F3.
Layout	Cadre commun (navbar + footer) partagé par plusieurs pages.
Composer	Gestionnaire de paquets PHP qui installe les librairies dans `vendor/`.
.htaccess	Configuration Apache qui redirige vers `index.php` (URLs propres).
Cache de templates	F3 compile les templates dans `tmp/`. Si une vue ne bouge pas : supprimer `tmp/*.php`.
Chemin absolu vs relatif	Absolu = commence par `/`. Relatif = dépend de l'URL, peut casser avec les routes.
DEBUG=3	Mode debug F3 : affiche les erreurs. En prod : `DEBUG=0`.