

Développement front

MMI 3 – TP#4 S5

Danielo **JEAN-LOUIS**

Nodejs

- Outil permettant l'utilisation du javascript côté serveur
 - Utilisation des mêmes fonctions sauf celles manipulant une page
 - Accès au système : dossiers, fichiers...
 - Basé sur le moteur js de Chrome : V8

Nodejs

- Eco-système vaste ayant permis l'émergence d'outils variés et utiles pour les développeurs front
 - Création d'application natives
 - Système d'exploitation
 - **Bundlers**
 - ...

Bundlers

- Outils nécessitant nodejs pour fonctionner
- Améliorent l'environnement de développement front-end
 - Indispensables de nos jours

Source(s) :

- <https://snipcart.com/blog/javascript-module-bundler>

Bundlers

- Optimisent les ressources (de tout types)
- Peuvent éliminer le code non utilisé
- Permettent d'utiliser du code non compatible pour le navigateur en temps normal
- Corrigent le problème d'interdépendances entre les fichiers js

Source(s) :

- <https://snipcart.com/blog/javascript-module-bundler>

Bundlers

- A la fin compilent les assets dans des formats compréhensibles par un navigateur
 - Tous les bundlers fonctionnent de cette façon

Source(s) :

- <https://snipcart.com/blog/javascript-module-bundler>

Bundlers

```
● ● ●  
  
<script src="script1.js" defer></script>  
<script src="script2.js" defer></script>  
<script src="script3.js" defer></script>
```

Chargement de scripts multiples. Attention à leur ordre



```
● ● ●  
  
<!-- Contient tous les scripts -->  
<script src="script.js" defer></script>
```

Généré par un bundler, il respecte l'ordre des dépendances et les contient toutes

Bundlers

- Importent des fichiers en tout genre (en fonction de la configuration)
- Importent des **modules**

Source(s) :

- <https://snipcart.com/blog/javascript-module-bundler>

Module javascript

- Se base sur la programmation modulaire
 - Découpage du code en unités autonomes
 - Limite le code spaghetti
- Peut exporter plusieurs modules
 - Fonctions, constantes, classes...
- Peut importer plusieurs modules

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/export>
- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/import>

Modules javascript

```
import myModule from "./my-module.js"
```

Import du module par défaut du fichier my-module

```
const myFunction () => { /**/ }  
export default myFunction // import myFunction from "..."  
  
const myFunction2 () => { /**/ }  
export myFunction2 // import { myFunction2 } from "..."
```

Exports de modules, un par défaut et un autre nommé

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/export>
- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/import>

Liste de bundlers (non exhaustive)

- Browserify (l'un des pionniers) - désuet
- Grunt / Gulp (gestionnaires de tâches) - désuet
- Webpack
- Rollup
- Parcel
- ...
- **Vite**

Source(s) :

- <https://snipcart.com/blog/javascript-module-bundler>

Vite

- (Autre) Bundler
- Fonctionne avec Nodejs
- Pensé pour le développement
- Se greffe à Rollup (autre bundler)
- Créée par Evan You, créateur de VueJS

Source(s) :

- <https://vitejs.dev/>

Vite

- Fonctionne clé en main
 - `npm create vite@latest`
- Dernière version majeure en date : v4
- Nécessite très peu de configuration par défaut

Source(s) :

- <https://vitejs.dev/>

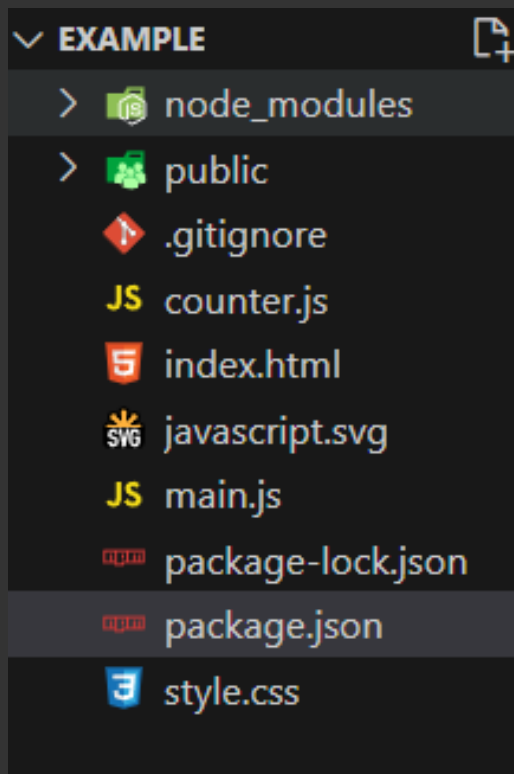
Vite

- Gère plusieurs frameworks js : Angular, React...
 - Via plugins
- Conforme aux derniers standards javascript
- Gère les ressources **de tout type dans le javascript**
 - Via plugins

Source(s) :

- <https://vitejs.dev/>

Vite

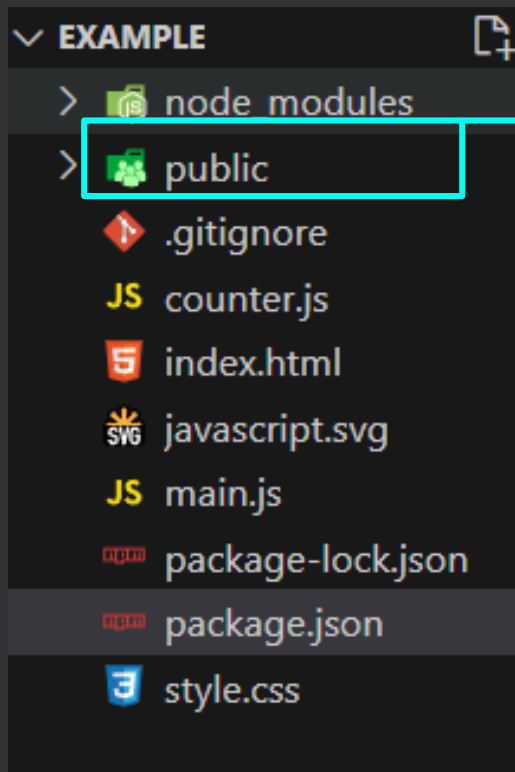


Après avoir installé les dépendances (npm install), nous sommes prêts à travailler (npm run dev)

Source(s) :

- <https://vitejs.dev/>

Vite



Contient les dépendances externes. Fichiers qui ne seront pas gérés par vite

Source(s) :

- <https://vitejs.dev/>

Vite – Dossier public/

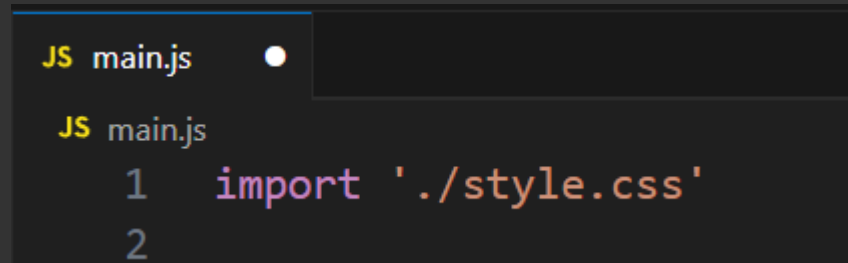
- Contient toutes les ressources qui n'ont pas besoin d'être gérées par vite
 - Ex : plugin javascript
- Chemin doit être absolu et sans "public/" dans le chemin
 - Ex : "public/icon.png" → "/icon.png"

Source(s) :

- <https://vitejs.dev/>
- <https://vitejs.dev/guide/assets.html#the-public-directory>

Vite – Différences avec l'existant

- Gestion du CSS dans le javascript
 - **On importe le CSS dans nos fichiers javascript**

A screenshot of a code editor with a dark theme. The top tab is labeled 'JS main.js' with a white dot indicating it is the active file. The editor area shows the same tab label 'JS main.js' followed by two lines of code: '1 import './style.css'' and '2'. The code is syntax-highlighted, with 'import' in purple and the string './style.css' in orange.

```
JS main.js
JS main.js
1  import './style.css'
2
```

Source(s) :

- <https://vitejs.dev/>

Pratiquons ! - Vite (Partie 1)

Pré-requis :

- Avoir la ressource ressources/vite

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-front-s5%2Ftravaux-pratiques%2Fnumero-4%2Fressources>

Vite

- Extensible via un système de plugins
 - Rajoute de nouvelles fonctionnalités et nouveaux types d'imports dans les fichiers javascript comme les **préprocesseurs CSS**

Source(s) :

- <https://github.com/vitejs/awesome-vite#plugins>

Préprocesseurs CSS

- Méta-langages CSS
- Ne sont pas lus par les navigateurs
- **Doivent être compilés en CSS**
- Permettent de simplifier l'écriture du CSS
- SCSS est le plus utilisé

Source(s) :

- <https://www.alsacreations.com/article/lire/1717-les-preprocesseurs-css-c-est-sensass.html>
- <https://grafikart.fr/tutoriels/differences-sass-scss-329>
- <https://la-cascade.io/se-lancer-dans-sass/>
- <https://sass-lang.com/>

Préprocesseurs CSS

- Utilisent une syntaxe proche du CSS
- Apportent de nouvelles fonctionnalités
 - Imbrication de sélecteurs
 - Limite la répétition de code
 - Conditions / boucles
 - **Variables compilées** – Elles ne sont pas modifiables après coup
 - ...

Préprocesseurs CSS - Exemple

```
✓ .conteneur-boite {  
  background-color: ■ red;  
  padding: 0.8rem;  
  
  .titre {  
    font-size: 1.5rem;  
  }  
  // ...  
}
```

Code SCSS



```
.conteneur-boite {  
  background-color: ■ red;  
  padding: 0.8rem;  
}  
  
.conteneur-boite .titre {  
  font-size: 1.5rem;  
}
```

Code CSS
(Une fois compilé)

Préprocesseurs CSS

- SCSS fonctionne avec Vite dès l'installation de SCSS dans le projet

A terminal window with a dark purple background and a light purple border. It features three small gray circles in the top-left corner, representing window control buttons. The command 'npm install -D sass' is displayed in a white monospaced font.

```
npm install -D sass
```

Une fois installé, vous pouvez importer des fichiers .scss

Pratiquons ! - Vite (Partie 2)

Pré-requis :

- Avoir la ressource ressources/vite

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopment-front-s5%2Ftravaux-pratiques%2Fnumero-4%2Fressources>

PostCSS

- Package ajoutant de nouvelles fonctionnalités à CSS
 - Nesting, mixins, lint...
- Se rapproche du CSS en terme de syntaxe
- Extensible via des plugins
- Géré nativement par vite

Source(s) :

- <https://github.com/postcss/postcss>

Moteur de templating

- Génère des pages HTML à partir de données
- Permet de respecter le V du modèle MVC
 - Limite le code spaghetti
- Langages différents du HTML
 - Doivent être compilés en HTML

Moteur de templating

- Propose moult fonctionnalités
- Souvent affecté à un framework...
 - Symfony (php) → **twig**
 - Django (python) → jinja
 - ror (ruby) → erb
 - ReactJS (js) → jsx
 - ...
- ...mais peut fonctionner sans (framework)

Moteur de templating

```
...  
<!-- template -->  
<p>Je suis étudiant en {{ formation }}</p>
```

Template

Moteur de template

```
...  
<!-- Données -->  
{ "formation": "MMI" }
```

Données

```
...  
<!-- HTML -->  
<p>Je suis étudiant en MMI </p>
```

HTML
(Compilé par le moteur)

Moteur de templating - PHP

- PHP est techniquement un moteur de templating mais possède de nombreux problèmes :
 - Mélange difforme entre PHP et HTML
 - Non-respect du modèle MVC
 - Failles de sécurité
 - Compliqué à maintenir

Twig

- Moteur de templating associé à Symfony
 - Permet aux développeurs backend de réutiliser votre code Twig
- Utilise la même syntaxe que jinja ou encore nunjucks
 - En connaître un, fait apprendre les trois

Source(s) :

- <https://twig.symfony.com/>
- <https://vituum.dev/plugins/twig.html>

Twig

- Utilisable avec Vite via un plugin
 - On utilisera @vituum/vite-plugin-twig
- Propose une syntaxe claire et facile à apprendre
- Extension de fichiers en .twig

Source(s) :

- <https://twig.symfony.com/>

Twig

```
• • •  
  
<?php  
foreach ($items as $value) {  
    if ($value.active) {  
?>  
          
  
<?php  
    }  
}  
?>
```

← Code PHP

Code twig →

```
• • •  
  
{% for value in items if value.active %}  
      
{% endfor %}
```

Source(s) :

- <https://twig.symfony.com/>

Twig - Syntaxe

- Trois syntaxes :
 - `{% __mot_clé__ %}` : fait quelque chose
 - Boucle, condition...
 - `{{ __mot_clé__ }}` : affiche quelque chose
 - `{# __mot_clé__ #}` : commentaire

Source(s) :

- <https://twig.symfony.com/>

Twig - Boucle

```

<ul>
    {% for user in users %}
        <li>{{ user.username }}</li>
    {% endfor %}
</ul>

```

Ici on parcourt un tableau “users” contenant des objets dont on accède à la clé “username” et on affiche le contenu dans la balise

Source(s) :

- <https://twig.symfony.com/>

Twig - Include

```
● ● ●  
  
{# file "index.twig" #}  
<div>  
    {% include 'includes/header.twig' %}  
</div>
```

Ici on inclus le contenu du fichier includes/header.twig dans le template "index.twig"

Source(s) :

- <https://twig.symfony.com/>

Twig - Block

- Permet de créer des “trous” dans un template pour qu’ils soient remplis par un autre template
- Fonctionne par héritage
 - Un template enfant ne “rempli” que les trous de son parent
- Fonctionne de pair avec le mot-clé “extends”

Source(s) :

- <https://twig.symfony.com/>

Twig - block

```
...  
{# parent.twig #}  
<head>  
    <link rel="stylesheet" href="style.css"/>  
    <title>{% block title %}{% endblock %} - Ma page</title>  
</head>
```

← Gabarit de référence

Gabarit enfant hérite de la structure du parent →

```
...  
{# enfant.twig #}  
{% extends "parent.html" %}  
{% block title %} Accueil {% endblock %}
```

Source(s) :

- <https://twig.symfony.com/>

Twig - block

- Un bloc “enfant” crée une page entière la compilation
- Les gabarits peuvent être réutilisés et servir de page
- Un block peut contenir une valeur par défaut
- Un block peut être dans une boucle

Source(s) :

- <https://twig.symfony.com/>

Templating avec Vite

- Nécessite un plugin dédié et l'utilisation d'un fichier de configuration (vite.config.js)
- Requiert une nomenclature très spécifique
 - La configuration peut être très fastidieuse si ce n'est pas respecté (voir sources)

Source(s) :

- <https://vituum.dev/guide/features.html#project-structure>

Templating avec Vite

- **public** - place for static files and dist files
- **src**
 - **assets** - your static files as `.png` , `.svg`
 - **data** - your `.json` data used in templates
 - **emails** - your email template files
 - **scripts** - your script files as `.js` , `.ts`
 - **styles** - your styles files as `.css` , `.scss`
 - **components** - your template files as `.twig` , `.latte`
 - **layouts** - your template layout files as `.twig` , `.latte`
 - **pages** - your pages as `.html` , you can also nest or define page as `.json` or `.twig` , `.latte` and other template engines

Voici la nomenclature recommandée. La modifier nécessitera des modifications laborieuse dans la configuration de vite

Source(s) :

- <https://vituum.dev/guide/features.html#project-structure>

vite.config.js

- Fichier servant de configuration pour vite
- Liste les plugins nécessaires au projet
- Placé par défaut à la racine du projet
- Nécessaire quand on modifie la structure de base du projet

Pratiquons ! - Vite (Partie 3)

Pré-requis :

- Avoir la ressource `ressources/vite-twig`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopment-front-s5%2Ftravaux-pratiques%2Fnumero-4%2Fressources>

Twig & Vite & JSON

- Injection de données json
 - Données doivent être dans le dossier src/data
 - Avoir l'extension .twig.json + nom template
 - Ex : index.twig → index.twig.json (à côté)
- Les données json sont statiques
 - Elles sont écrites en dur au final dans le fichier HTML

Source(s) :

- <https://twig.symfony.com/>


Twig & Vite & JSON

- Plusieurs fichiers peuvent être importés en même temps, mais attention aux noms de clés si elles sont semblables entre les fichiers, vite tranchera

Source(s) :

- <https://twig.symfony.com/>

Twig & Vite & JSON



```
{% for item in list %}  
    <li>  
        {{ item.name }}  
    </li>  
{% endfor %}
```

On parcourt un tableau d'objets (list) et pour chaque objet on affiche la clé "name"

Source(s) :

- <https://twig.symfony.com/>

Pratiquons ! - Vite (Partie 4)

Pré-requis :

- Avoir la ressource `ressources/vite-twig`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-front-s5%2Ftravaux-pratiques%2Fnumero-4%2Fressources>

Twig & Vite & JSON

- Création de page depuis un fichier JSON grâce à la clé “template”



```
// my-page.json
{
  "template": "layouts/main.twig",
  "title": "100 % JSON"
}
```

Lorsqu'on accède à /my-page, on charge le template “layouts/main.twig” et on injecte la variable “title” dans le template Twig

Source(s) :

- <https://vituum.dev/guide/template-engines.html#template-engines>
- <https://stackblitz.com/github/vituum/vituum/tree/main/examples/twig?file=package.json>

Twig & Vite & JSON

- La page html est créée lors de la compilation
- Le template chargé doit être à l'extérieur du dossier "pages/"

Source(s) :

- <https://vituum.dev/guide/template-engines.html#template-engines>
- <https://stackblitz.com/github/vituum/vituum/tree/main/examples/twig?file=package.json>

Variables d'environnement (env vars)

- Permettent d'injecter du contenu statique dans nos fichiers
 - Par exemple : URL de serveur d'API
- Ne modifie pas le code source
 - Limite le risque de bugs, duplication de code et modifications inutiles dans un commit

Variables d'environnement (env vars)

- Géré par défaut par vite
- Vite crée par défaut des env vars
 - `MODE` : production / developpement
 - `PROD` / `DEV` : booléen
 - ...

Source(s) :

- <https://vitejs.dev/guide/env-and-mode.html>

Variables d'environnement (env vars)

- Accessibles partout dans le projet (HTML/JS)
- Stockées dans un fichier .env
 - Possibilité d'en avoir un par env
 - .env.production, .env.developpement...

Source(s) :

- <https://vitejs.dev/guide/env-and-mode.html>

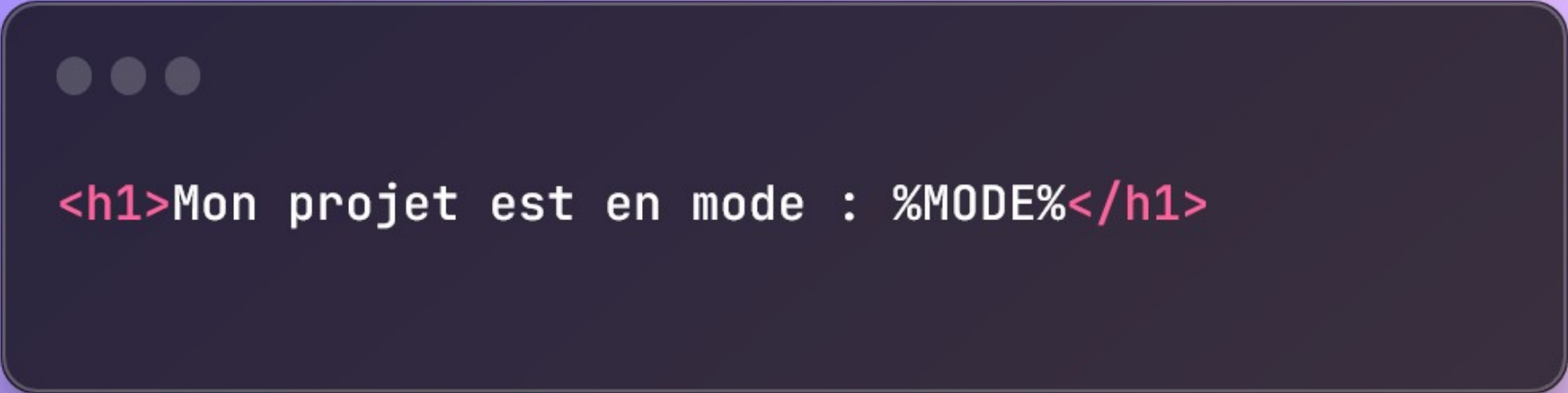
Variables d'environnement (env vars)

- **Le contenu est accessible à l'utilisateur final**
 - Éviter de mettre des données sensibles
 - Risque d'exposition dans le code source

Source(s) :

- <https://vitejs.dev/guide/env-and-mode.html>

Variables d'environnement (env vars)



A terminal window with a dark background and three light gray window control buttons in the top-left corner. The text inside the terminal is a line of HTML code: `<h1>Mon projet est en mode : %MODE%</h1>`. The opening and closing tags `<h1>` and `</h1>` are highlighted in red, while the text between them is white.

```
<h1>Mon projet est en mode : %MODE%</h1>
```

On affiche la variable d'environnement "MODE" dans notre template (ça fonctionne également en HTML)

Source(s) :

- <https://vitejs.dev/guide/env-and-mode.html>

Variables d'environnement (env vars)

A code editor window with a dark background and three window control buttons in the top-left corner. It contains a single line of JavaScript code: `console.log(import.meta.env.MODE)`. The code is color-coded: `console.log` is green, `import` is pink, `meta` is blue, `env` is light blue, and `MODE` is blue.

```
console.log(import.meta.env.MODE)
```

On affiche la variable d'environnement "MODE" dans notre fichier javascript

Source(s) :

- <https://vitejs.dev/guide/env-and-mode.html>

Point technique – env vars

- Créez un fichier de gabarit qui sera copié
 - Ex : `.env.dev` → `.env.dist.dev`
 - Le fichier `.env.dev` ne sera jamais commité
 - Le fichier `.env.dist.dev` sert de gabarit, il indique aux autres développeurs les variables attendues
- Évitez de commiter le fichier `.env` qui sera modifié régulièrement

Env vars & Vite

- Doivent être préfixées par “VITE_”
 - Sinon la variable ne sera pas accessible dans le navigateur
- Vite charge un fichier différent en fonction de la commande
 - `.env.developpement` → vite
 - `.env.production` → vite build

Source(s) :

- <https://vitejs.dev/guide/env-and-mode.html>

Env vars & Vite



```
# .env.local
```

```
# VITE_COMMENTE=pas injecté dans le code
```

```
VITE_CUSTOM_VAR=MMI
```

Notre fichier .env contient deux variables dont une commentée

Source(s) :

- <https://vitejs.dev/guide/env-and-mode.html>

Env vars & Vite



```
<p>J'affiche une variable : %VITE_CUSTOM_VAR%</p>
```

Même les variables d'env personnalisées sont injectées dans les fichiers

Source(s) :

- <https://vitejs.dev/guide/env-and-mode.html>

Pratiquons ! - Vite (Partie 5)

Pré-requis :

- Avoir la ressource `ressources/vite-twig`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-front-s5%2Ftravaux-pratiques%2Fnumero-4%2Fressources>

vite build

- Accessible via la commande `npm run build`
- Permet de réaliser une version pour la production du projet
 - Optimisation des assets
 - Compilation des templates en HTML
 - ...
- **Crée des chemins absolus**

vite build – Chemins absolus

- Rend compliqué la mise en ligne du projet dans un sous-dossier
- Utilisation de la clé "base" dans le fichier de configuration pour définir l'url de base des fichiers
 - Sous MacOS/linux, vous pouvez utiliser ./

Source(s) :

- <https://vitejs.dev/config/shared-options.html#base>

Conclusion

- Utilisation d'outil comme Vite est indispensable pour le développement front moderne
- L'utilisation de moteur de templates améliore le développement HTML et le rend plus simple
 - Permet une réutilisation simple du code pour les développeurs back-end en fonction du projet

Conclusion

- Même si Vite venait à disparaître demain, son successeur fonctionnera plus ou moins de la même façon
 - Utilise nodejs (ou équivalent)
 - Compile les dépendances en un fichier js
 - Améliore l'expérience de développement front
 - Hot reload
 - Serveur de développement
 - ...

Questions ?

