

# 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
  - Nouvelles fonctions permettant l'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 tous types)
- Corrigent le problème d'interdépendances entre les fichiers js
- Peuvent éliminer le code non utilisé

Source(s) :

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

# Bundlers

- 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 dans le javascript
  - 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) - désuets
- Webpack
- Rollup
- Parcel
- ...
- **Vite**

Source(s) :

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

# Vite

- (Autre) Bundler
- Fonctionne avec Nodejs
- Se greffe à Rollup (autre bundler)
- Créé 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 : v5 (2024)
- Nécessite très peu de configuration par défaut
- Actualise le navigateur après chaque changement
  - Plus besoin de recharger la page soi-même

Source(s) :

- <https://vitejs.dev/>

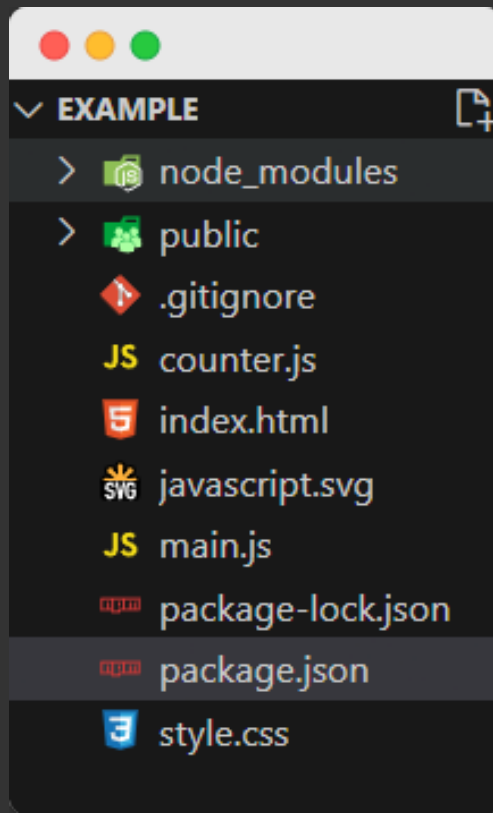
# Vite

- Gère les frameworks js : Angular, React...
  - Via plugins
- Supporte le code-splitting
- Importe les ressources **de tout type dans le javascript**
  - Via plugins

Source(s) :

- <https://vitejs.dev/>

# Vite



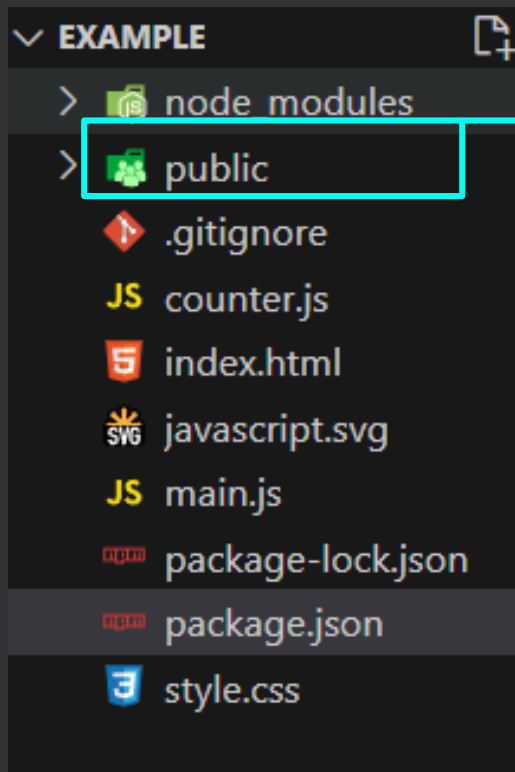
Après avoir installé les dépendances (npm install), nous sommes prêts à travailler (npm run dev)

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/

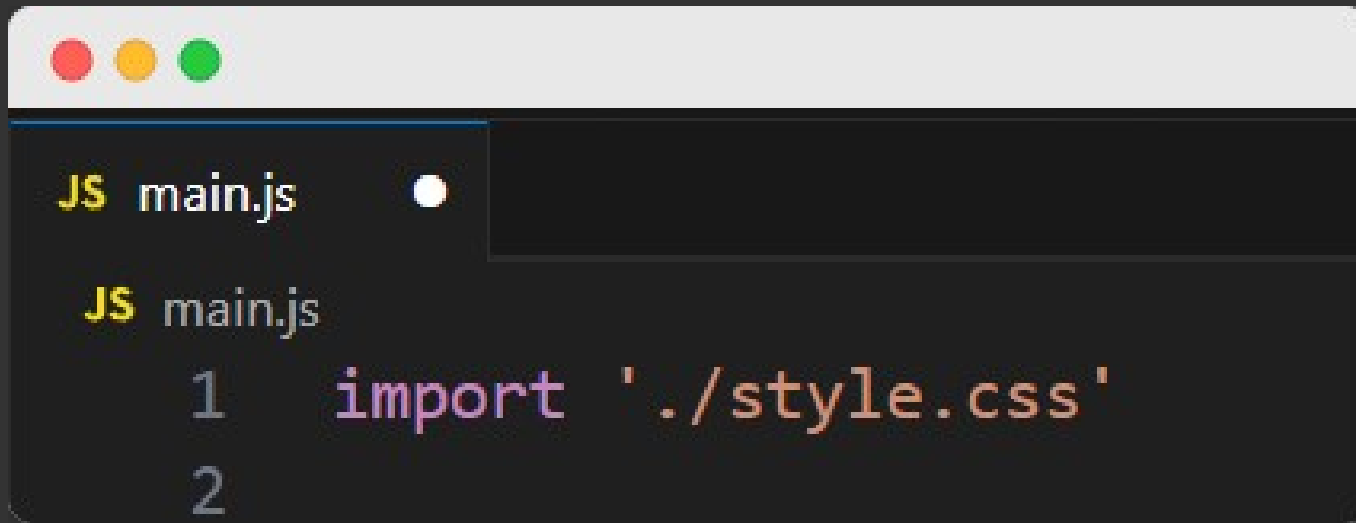
- Contient toutes les ressources qui n'ont pas besoin d'être gérées par vite
  - Ex : reset.css, favicon...
- Chemin doit être absolu et sans "public/" dedans
  - Ex : "public/icon.png" → "/icon.png"

## 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**



Source(s) :

- <https://vitejs.dev/>

# Vite – Commandes

- vite : lance le serveur de développement
  - Gère la compilation ainsi que le rechargement automatique de la page
- vite build : compile les fichiers pour la production
  - Crée un dossier dist/ qui contient tout votre projet. **Ne pas commiter ce dossier**

Source(s) :

- <https://vitejs.dev/guide/cli.html>

# Vite – Commandes

- vite preview : **simule** un serveur de prod
  - **Ne pas utiliser sur un serveur de production**
  - Nécessite d'avoir un build avant

Source(s) :

- <https://vitejs.dev/guide/cli.html>

# Pratiquons ! - Vite (Partie 1)

Pré-requis :

- Avoir la ressource ressources/vite

A télécharger ici :

[https://github.com/DanYellow/cours/raw/refs/heads/main/developement-front-s5/travaux-pratiques/numero-4/developement-front-s5\\_travaux-pratiques\\_numero-4.ressources.zip](https://github.com/DanYellow/cours/raw/refs/heads/main/developement-front-s5/travaux-pratiques/numero-4/developement-front-s5_travaux-pratiques_numero-4.ressources.zip)

# 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 populaire

Source(s) :

- <https://www.alsacreations.com/article/lire/1717-les-preprocesseurs-css-c-est-sensass.html>
- [https://developer.mozilla.org/fr/docs/Glossary/CSS\\_preprocessor](https://developer.mozilla.org/fr/docs/Glossary/CSS_preprocessor)



# 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 dans le CSS
  - ...

Source(s) :

- <https://www.alsacreations.com/article/lire/1717-les-preprocesseurs-css-c-est-sensass.html>
- [https://developer.mozilla.org/fr/docs/Glossary/CSS\\_preprocessor](https://developer.mozilla.org/fr/docs/Glossary/CSS_preprocessor)

# Préprocesseurs CSS

- **Ne jamais éditer le fichier CSS compilé**
  - Les modifications seront écrasées à la modification du fichier source

## Source(s) :

- <https://www.alsacreations.com/article/lire/1717-les-preprocesseurs-css-c-est-sensass.html>
- [https://developer.mozilla.org/fr/docs/Glossary/CSS\\_preprocessor](https://developer.mozilla.org/fr/docs/Glossary/CSS_preprocessor)

# Préprocesseurs CSS – Exemple SCSS

```
...  
  
.container {  
  background-color: red;  
  padding: 0.8rem;  
  .title {  
    font-size: 1.35rem;  
  }  
  // [...]  
}
```

↑  
Code SCSS

Le code CSS, une fois compilé

```
...  
  
.container {  
  background-color: red;  
  padding: 0.8rem;  
  /* [...] */  
}  
  
.container .title {  
  font-size: 1.35rem;  
}
```

Source(s) :

- <https://grafikart.fr/tutoriels/differences-sass-scss-329>
- <https://la-cascade.io/se-lancer-dans-sass/>
- <https://sass-lang.com/>

# Préprocesseurs CSS – SCSS et Vite

- 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 has three small circles in the top left corner. The text 'npm install -D sass' is displayed in a white monospace font.

```
npm install -D sass
```

Une fois installé, vous pouvez importer des fichiers .scss dans le javascript

Source(s) :

- <https://vitejs.dev/guide/features.html#css-pre-processors>

# Préprocesseurs CSS

- Fonctionnalité d'imbrication est disponible et bien gérée nativement en CSS maintenant (CSS Nesting – voir source)

## Source(s) :

- [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_nesting/Using\\_CSS\\_nesting](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_nesting/Using_CSS_nesting)

# Pratiquons ! - Vite (Partie 2)

Pré-requis :

- Avoir la ressource ressources/vite

A télécharger ici :

[https://github.com/DanYellow/cours/raw/refs/heads/main/developement-front-s5/travaux-pratiques/numero-4/developement-front-s5\\_travaux-pratiques\\_numero-4.ressources.zip](https://github.com/DanYellow/cours/raw/refs/heads/main/developement-front-s5/travaux-pratiques/numero-4/developement-front-s5_travaux-pratiques_numero-4.ressources.zip)

# PostCSS

- Package ajoutant de nouvelles fonctionnalités à CSS
  - Nesting, mixins, linter...
- Se rapproche du CSS en terme de syntaxe
- Extensible via des plugins
- Géré nativement par Vite
- Utilisation **recommandée** par Vite

Source(s) :

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

# CSS Nesting / Imbrication de style

- Alternative aux préprocesseurs CSS
  - Utilise la même syntaxe plus ou moins
  - Permet l'imbrication de sélecteurs CSS notamment
- Nouveauté CSS (apparu fin 2023)
  - Non géré par les anciens navigateurs

## Source(s) :

- [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_nesting/Using\\_CSS\\_nesting](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_nesting/Using_CSS_nesting) - anglais
- <https://genesis-technology.fr/pourquoi-le-css-nesting-module-est-une-revolution-pour-le-frontend/>



# CSS Nesting - Exemple

```
.navigation {  
  display: flex;  
  .navigation-el {  
    font-size: 1.25rem;  
    color: blue;  
  }  
}
```

L'imbrication est interprétée de la façon suivante par le navigateur

```
.navigation {  
  display: flex;  
}  
.navigation .navigation-el {  
  font-size: 1.25rem;  
  color: blue;  
}
```

## Source(s) :

- [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_nesting/Using\\_CSS\\_nesting](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_nesting/Using_CSS_nesting) - anglais
- <https://genesis-technology.fr/pourquoi-le-css-nesting-module-est-une-revolution-pour-le-frontend/>

# CSS Nesting - Exemple

```
● ● ●  
  
.paragraphe {  
  color: red;  
  
  &:hover {  
    color: blue;  
  }  
}
```

Dans le cas d'une pseudo-classe, on la préfixe d'une esperluette "&". Fonctionne également avec les pseudo-elements et les classes

```
● ● ●  
  
.paragraphe {  
  color: red;  
}  
  
.paragraphe:hover {  
  color: blue;  
}
```

## Source(s) :

- [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_nesting/Using\\_CSS\\_nesting](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_nesting/Using_CSS_nesting) - anglais
- <https://genesis-technology.fr/pourquoi-le-css-nesting-module-est-une-revolution-pour-le-frontend/>

# 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 (possible) du modèle MVC
  - Failles de sécurité
  - Compliqué à maintenir

# Nunjucks

- Moteur de templating
- Utilise la même syntaxe que jinja ou encore twig
  - En connaître un, fait apprendre les trois
  - Permet aux développeurs symfony ou jinja de récupérer votre code

Source(s) :

- <https://mozilla.github.io/nunjucks/templating.html>
- <https://vituum.dev/plugins/nunjucks.html>

# Nunjucks

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

Source(s) :

- <https://mozilla.github.io/nunjucks/templating.html>



# Nunjucks

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

← Code PHP

Code nunjucks →

```
...  
{% for item in items %}  
    {% if item.active %}  
          
    {% endif %}  
{% endfor %}
```

# Nunjucks - Syntaxe

- Trois syntaxes :
  - `{% __contenu__ %}` : fait quelque chose
    - Boucle, condition...
  - `{{ __contenu__ }}` : affiche quelque chose
  - `{# __contenu__ #}` : commentaire
    - Note : les commentaires ne sont pas visibles dans le navigateur

Source(s) :

- <https://mozilla.github.io/nunjucks/templating.html>

# Nunjucks - Boucle

```

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

```


On parcourt un tableau “users” contenant des objets dont on accède à la clé “username” et on affiche le contenu dans une balise <li>

Note : les boucles fonctionnent également avec les objets

Source(s) :

- <https://mozilla.github.io/nunjucks/templating.html#for>

# Nunjucks - Include



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

On inclut le contenu du fichier `includes/header.njk` dans le template `"index.njk"`

Note : ça fonctionne également avec un fichier `html`

Source(s) :

- <https://mozilla.github.io/nunjucks/templating.html#include>

# Nunjucks - Bloc

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

Source(s) :

- <https://mozilla.github.io/nunjucks/templating.html>

# Nunjucks - Bloc

← Gabarit de référence

```
...  
{# file "parent.njk" #}  
<head>  
  {# [...] #}  
  <link rel="stylesheet" href="/assets/css/fonts.css">  
  <title>{% block title %}{% endblock %} - Mon super site</title>  
</head>
```

Gabarit enfant hérite de la structure du parent et remplit ses *blocks* →

```
...  
{# file "enfant.njk" #}  
{% extends "layouts/parent.njk" %}  
{% block title %} Accueil {% endblock %}
```

Source(s) :

- <https://mozilla.github.io/nunjucks/templating.html#block>

# Nunjucks - Bloc

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

Source(s) :

- <https://mozilla.github.io/nunjucks/templating.html#block>

# Nunjucks - Macro

- Équivalent des fonctions dans n'importe langage de programmation
  - Possibilité passer des arguments avec/ou non valeurs par défaut
  - Réutilisable

Source(s) :

- <https://mozilla.github.io/nunjucks/templating.html#macro>




# Nunjucks - Macro

- Un fichier peut contenir plusieurs macros
- Par défaut, la macro est scopée
  - Pas d'accès aux variables extérieures

Source(s) :

- <https://mozilla.github.io/nunjucks/templating.html#macro>

# Nunjucks - Macro



```
{% macro link(label, url='', open_new_tab=false) %}  
<a  
  href="{{ url }}"  
  {{ target="_blank" if open_new_tab == true else "" }}  
  rel="noopener noreferrer"  
>  
  {{ label }}  
</a>  
{% endmacro %}
```

On définit une macro nommée “link”. Il suffit d’importer le fichier contenant la macro où vous en avez besoin

Source(s) :

- <https://mozilla.github.io/nunjucks/templating.html#macro>

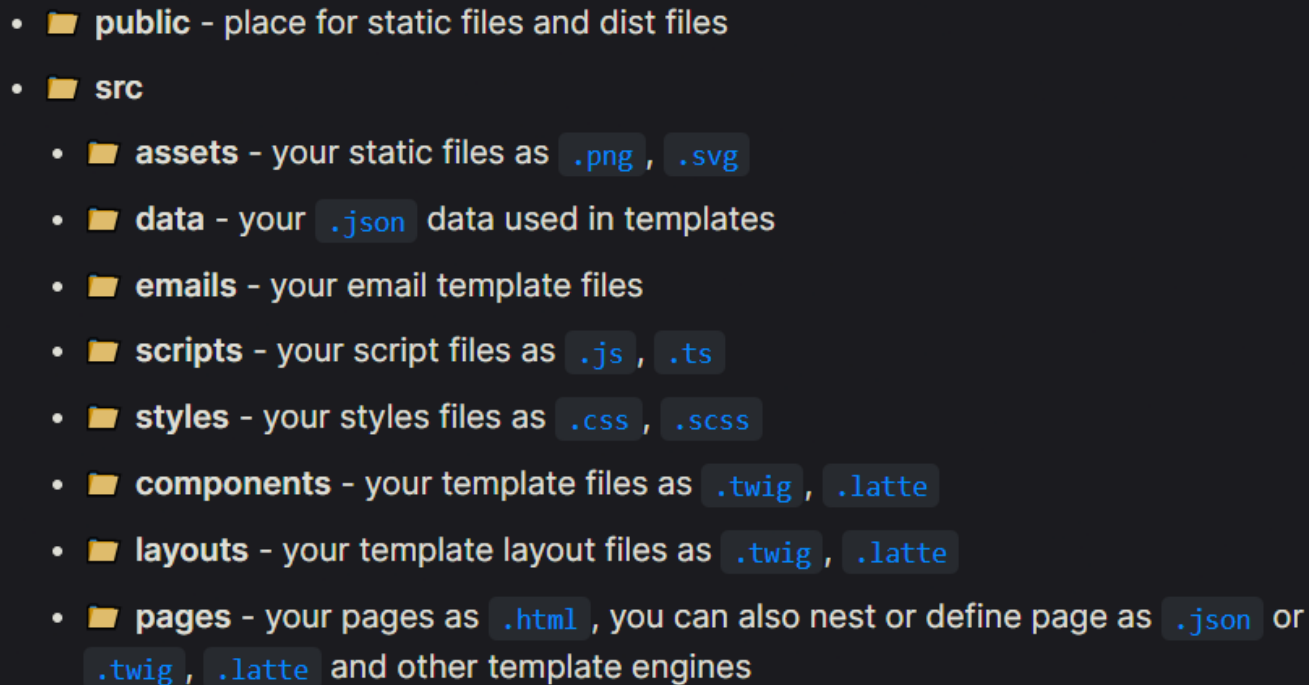










# Templating avec Vite

- Nécessite un plugin dédié et l'utilisation d'un fichier de configuration (vite.config.js)
- Au build, les templates sont compilés en HTML
  - Impossibilité d'avoir des données dynamiques
- 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 laborieuses 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 configuration de base ou ajoute des plugins

# Pratiquons ! - Vite (Partie 3)

Pré-requis :

- Avoir la ressource ressources/vite

A télécharger ici :

[https://github.com/DanYellow/cours/raw/refs/heads/main/developement-front-s5/travaux-pratiques/numero-4/developement-front-s5\\_travaux-pratiques\\_numero-4.ressources.zip](https://github.com/DanYellow/cours/raw/refs/heads/main/developement-front-s5/travaux-pratiques/numero-4/developement-front-s5_travaux-pratiques_numero-4.ressources.zip)

# Nunjucks & Vite & JSON


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

# Nunjucks & Vite & JSON

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



# Nunjucks & 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"

# Nunjucks & Vite & JSON

- **Attention** : Le fichier JSON doit contenir des clés à la racine
  - Il ne peut pas contenir qu'un tableau

```
[{"title": "MMI"}, {"title": "SRC"}]
```



**Interdit**

Le contenu n'est pas dans une clé

**Correct**

Le contenu est dans une clé



```
{  
  "data": [{"title": "MMI"}, {"title": "SRC"}]  
}
```

# Pratiquons ! - Vite (Partie 4)

Pré-requis :

- Avoir la ressource ressources/vite-njk

A télécharger ici :

[https://github.com/DanYellow/cours/raw/refs/heads/main/developement-front-s5/travaux-pratiques/numero-4/developement-front-s5\\_travaux-pratiques\\_numero-4.ressources.zip](https://github.com/DanYellow/cours/raw/refs/heads/main/developement-front-s5/travaux-pratiques/numero-4/developement-front-s5_travaux-pratiques_numero-4.ressources.zip)

# Nunjucks & Vite & JSON

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

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

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

Source(s) :

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

# Nunjucks & Vite & JSON

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

## Source(s) :

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

# Variable d'environnement (env var)

- Initié par la méthode “Twelve-Factor App Methodology”
  - Méthode visant à produire des logiciels de meilleure qualité
    - <https://12factor.net/fr/config>
- Contenu dans un fichier externe
  - Un fichier peut contenir plusieurs variables

Source(s) :

- <https://hyperlane.co/blog/the-benefits-of-environment-variables-and-how-to-use-them> - anglais

# Variable d'environnement (env var)

- Permet de stocker des données (sensibles)
  - Par exemple : URL de serveur d'API, mdp
- Ne modifie pas le code source
  - Limite le risque de bugs, duplication de code et modifications inutiles dans un commit

## Source(s) :

- <https://hyperlane.co/blog/the-benefits-of-environment-variables-and-how-to-use-them> - anglais

# Variable d'environnement (env var)

- Augmente la sécurité du projet
  - Les données ne sont pas accessibles sur un VCS
- Concept transverse à plein de langages
- 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>
- <https://jeremiechazelle.dev/gerer-les-variables-denvironnement-dans-un-projet-node-js/>



# Variable d'environnement (env var)

- **Le contenu est accessible à l'utilisateur final**
  - Éviter de mettre des données sensibles
    - Risque d'exposition dans le code source
- Les variables sont interprétées comme étant des chaînes de caractères
  - Il faudra faire les conversions dans votre code
- Géré par défaut par vite

Source(s) :

- <https://vitejs.dev/guide/env-and-mode.html>
- <https://jeremiechazelle.dev/gerer-les-variables-denvironnement-dans-un-projet-node-js/>

# Variable d'environnement (env var)

```
# .env
```

```
API_URL="http://api.example.com"
```

```
SSO_LOGIN_URL=http://sso.example.com
```

```
ENABLE_FEATURE=delete_card, active_pay
```

Exemple de fichier .env avec trois variables env

Source(s) :

- <https://vitejs.dev/guide/env-and-mode.html>
- <https://jeremiechazelle.dev/gerer-les-variables-denvironnement-dans-un-projet-node-js/>

## Point technique – env vars

- Créez un fichier de gabarit qui sera copié par les autres développeurs
  - 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

## Point technique – env vars

- Évitez de commiter le fichier .env :
  - Il sera modifié régulièrement
  - Peut contenir des données sensibles
- Utilisez le fichier .gitignore pour exclure les fichiers et dossiers qui ne doivent pas être commités

# Env vars & Vite

- Vite crée par défaut des env vars
  - MODE : production / développement
  - PROD / DEV : booléen (définit si on est en dev)
  - ...
- Accessible partout dans le projet (HTML/JS)

Source(s) :

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

# 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 (#)

(Note : il n'est pas utile de mettre des guillemets)

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>



# Env vars & Vite



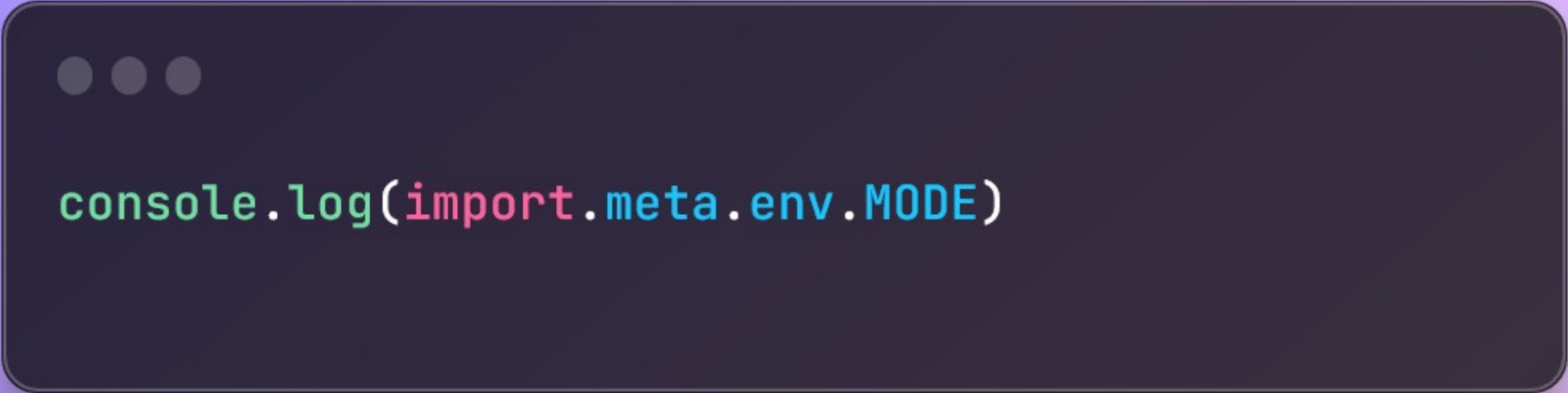
```
<h1>Mon projet est en mode : %MODE%</h1>
```

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>

# Env vars & Vite



```
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>

# Pratiquons ! - Vite (Partie 5)

Pré-requis :

- Avoir la ressource ressources/vite-njk

A télécharger ici :

[https://github.com/DanYellow/cours/raw/refs/heads/main/developement-front-s5/travaux-pratiques/numero-4/developement-front-s5\\_travaux-pratiques\\_numero-4.ressources.zip](https://github.com/DanYellow/cours/raw/refs/heads/main/developement-front-s5/travaux-pratiques/numero-4/developement-front-s5_travaux-pratiques_numero-4.ressources.zip)

# 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

- Vite (ou autre) 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
    - Rechargement automatique de la page
    - Serveur de développement
    - ...

**Questions ?**



