

# INF5190 - MVC - Vue et contrôleur

Jean-Philippe Caissy

25 septembre 2019

# Vue

- ▶ Représentation de l'information retourné par le contrôleur
- ▶ Possibilité d'avoir plusieurs vues pour la même représentation des données
  - ▶ Exemple : HTML et JSON

# Vue

## Engin de templating

- ▶ Utiliser une méthode Python pour construire la vue n'est pas efficace :
  - ▶ Difficile à tester
  - ▶ Code difficile à réutiliser
  - ▶ Manque de flexibilité
  - ▶ Peut introduire des failles de sécurité

Solution : utiliser un outil de génération de pages HTML!

# Vue

## Engin de templating

### Jinja2

Jinja2 est un engin de templating HTML

Avantages :

- ▶ Compilation des templates dans un bac à sable (sandbox)
- ▶ Escaping des entrées HTML automatique pour prévenir des attaques de type cross site scripting (XSS)
- ▶ Héritage de thèmes : permet de réutiliser des pages HTML
- ▶ Syntaxe configurable et extensible
- ▶ Facile à débbugger

# Vue

## Engin de templating

### Jinja2

```
>>> from jinja2 import Template
>>> template = Template('Hello World, {{ name }}!')
>>> template.render(name='John Doe')
'Hello World, John Doe!'
```

# Vue

## Engin de templating

### Jinja2

Syntaxe de base :

- ▶ `{{ variable }}` Pour afficher une variable
- ▶ `{% %}` Bloc d'exécution
  - ▶ `{% if i > 10 %} [...] {% endif %}`
  - ▶ `{% for item in navigation %}`  
    `<li>`  
        `<a href="{{ item.href }}">{{ item.name }}</a>`  
    `</li>`  
    `{% endfor %}`

# Vue

## Engin de templating

### Jinja2

### Filtres

On peut appliquer des filtres sur des variables

- ▶ `{{ variable|lower }}` pour afficher variable en minuscules
- ▶ `{{ 123.5321|round }}` retourne 123
- ▶ etc ...

Liste des filtres inclus

# Vue

## Engin de templating

### Jinja2

#### Fichier view.html

```
{% extends "layout.html" %}  
{% block body %}  
    <h1>Hello World, {{ user }}!</h1>  
{% endblock %}
```

#### Fichier layout.html

```
<html>  
    <body>  
        {% block body %}{% endblock %}  
    </body>  
</html>
```



## Internationalisation et localisation

- ▶ Internationalisation (i18n) et localisation (l10n)
  - ▶ 18 représente le nombre de lettre en le I et le N
  - ▶ idem pour l10n : 10 lettre entre le L et le N

Moyens pour adapter un logiciel à différentes langues, particularités régionales et spécifications techniques d'un marché local.

# Contrôleur

## Internationalisation et localisation

Objectif : adapter le code d'une application pour qu'il puisse être adapté à un marché/région/localisation

- ▶ Extraction de contenu :
  - ▶ Chaînes de caractères pour traduction
  - ▶ Nombres
  - ▶ Dates
  - ▶ Valeurs monétaires
  - ▶ Unités de mesure
- ▶ Utiliser le bon contenu en fonction de la locale (langue et région) demandée

# Contrôleur

## Internationalisation et localisation

Paramètre régionaux : définitions de textes et de formats utiles pour régionaliser une application

- ▶ Habituellement composé d'un identificateur de langue et de région :
  - ▶ en\_CA : Anglais (English), Canadien
  - ▶ en\_US : Anglais (English), États-Unis
  - ▶ fr\_CA : Français, Canada
  - ▶ fr\_FR : Français, France
  - ▶ fr\_BE : Français, Belgique
  - ▶ es\_AR : Espagnol, Argentine
  - ▶ etc. . .

# Contrôleur

## Internationalisation et localisation

### Exemples

Paramètre régional	Valeur
en_US	123 456,789
fr_CA	123,456.789
-	-
en_US	9/25/19
fr_FR	25/09/2019
-	-
en_CA	\$123,456.79
fr_CA	123 456,79 \$

# Contrôleur

## Internationalisation et localisation

- ▶ L'extraction des chaînes de caractères permet de traduire et localiser selon la région demandé
- ▶ La région peut être incluse dans l'URL :
  - ▶ `/fr-ca/ressource/1`
  - ▶ `/en-us/ressource/1`
- ▶ Le navigateur envoie même les langues supportés par le système dans un entête :
  - ▶ `Accept-Language: en-CA,en-US;q=0.7,en;q=0.3`

# Liens

- ▶ Utilisation de i18n et l10n dans flask
- ▶