Auteur externe

Zeste de savoir

Votre site web avec Bailador !

12 août 2019

Table des matières

Ce tutoriel n'est plus à jour ! En effet, il porte sur la version 0.0.1 de Bailador. Le développement du framework est aujourd'hui très actif. Au 28 juin, Bailador était dans sa version 0.0.6. Il est donc possible que vous vous confrontiez à quelques problèmes en suivant ce tutoriel.
Toutefois, un nouveau contenu sur Bailador, beaucoup plus complet et à jour, devrait bientôt faire son apparition ! Vous pouvez suivre l'évolution de ce nouveau tutoriel sur GitHub .

Depuis plusieurs années maintenant, les sites Web ont gagné en fonctionnalités, et sont devenus dans le même temps de plus en plus complexes. Avec Bailador, vous aurez la possibilité de mettre en place une application rapidement et efficacement. Alors n'attendez plus, et faites le choix de la facilité !

Prérequis
Connaître les bases du langage Perl (dans sa version 6).
Prérequis optionnels
Connaissances à propos du Web.
Objectifs
Apprendre à créer un site Web.

1. Bailador et Perl 6

Connaissez-vous Perl ? « Seulement de nom » ? Cela ne m'étonne pas !

Assez méconnu de nos jours, notamment depuis la naissance et l'envol de PHP dans les années 1990, Perl est un langage de programmation à la réputation ésotérique. Certains poussent même le vice jusqu'à le qualifier de difficile à comprendre, lui donnant l'image d'un outil terrifiant au moyen duquel les « *nerds* » du côté obscur de la Force produisent des programmes cryptiques, que bien peu de courageux aventuriers tentent de modifier après leur passage. Programmer en Perl relèverait donc, dans l'imaginaire collectif, tant de l'exploit que de la magie noire…

Pourtant, et vous le découvrirez rapidement par vous même, Perl est un langage facile à utiliser, disposant d'une syntaxe claire et efficace, tout en restant extrêmement puissant. Un nombre conséquent d'applications et de services Web connus utilisent Perl, tels qu'Amazon , IMDb , slashdot , les serveurs Bugzilla , ou encore une partie du gestionnaire de version git.

Perl est un langage de programmation créé en 1987 par Larry Wall. À l'origine, il était surtout utilisé pour le développement de scripts d'administration système sous UNIX, mais, avec les années, et après plusieurs révisions importantes du langage, Perl est rapidement devenu un outil polyvalent, puissant et extrêmement pratique, ce qui lui vaut aujourd'hui le surnom humoristique de « rouleau de scotch de l'Internet ».

FIGURE 1. – Larry Wall.

Toutefois, Perl avait besoin de changements. C'est ainsi que depuis le début des années 2000, Larry Wall et une bande d'irréductibles programmeurs travaillent sur une toute nouvelle version du langage : Perl 6. L'une des particularités de Perl 6 est de ne pas disposer d'implémentation de référence. Chacun peut donc créer sa propre implémentation, à partir de nombreuses spécifications techniques disponibles. Au cours de ce tutoriel, nous utiliserons l'implémentation la plus répandue, **Rakudo Star**.

L'objectif de ce tutoriel est d'apprendre à créer des sites Web dynamiques, de manière simple et efficace. Pour cela, nous utiliserons donc, vous l'aurez compris, un micro-*framework*, Bailador. Bailador est en réalité une version pour Perl 6 d'un célèbre *framework* Perl : Dancer (Dancer, qui s'est lui même grandement inspiré du *framework* Sinatra de Ruby).

> **Pourquoi un micro-*framework* ?**
>
> Dancer, Sinatra et bien sûr Bailador sont considérés comme des micro-*frameworks*. Cela signifie simplement qu'ils n'englobent pas autant de fonctionnalités que des *frameworks* plus complets, comme Django (Python) ou Ruby on Rails (Ruby). Toutefois, ils savent tirer avantage de cette situation : il est possible de mettre en place une application très rapidement, et très facilement. Je ne vous ferai pas attendre plus longtemps : sans plus tarder, place à la pratique !

1.1. Installation des outils nécessaires

Comme indiqué plus haut, nous utiliserons **Rakudo Perl 6**. Vous trouverez sur cette page des informations complètes sur l'installation, et ce sous les différents systèmes d'exploitation.

Sous Windows, vous pouvez directement télécharger un exécutable, ici .

Sous OS X, Rakudo Star est disponible depuis Homebrew.

```
brew install rakudo-star
```

Sous Linux et les Unix-*like*, le moyen le plus simple de se procurer Rakudo est d'utiliser **rakudobrew**, qui rend l'installation plus facile.

```
git clone https://github.com/tadzik/rakudobrew ~/.rakudobrew
echo 'export PATH=~/.rakudobrew/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
```

Puis, pour installer Rakudo…

```
rakudobrew build moar
```

Vous disposez désormais de votre version de Perl 6 !

Il nous reste désormais à installer notre module, Bailador. Pour cela vous disposez de deux options. Premièrement, vous pouvez l'installer manuellement, depuis GitHub .

Deuxièmement, et plus facilement, vous pouvez aussi utiliser le gestionnaire de modules de Perl 6, panda . Les informations d'installation vous sont données sur ce dernier lien. Notez que si vous utilisez rakudobrew, vous pouvez installer panda très facilement.

```
rakudobrew build panda
```

Une fois panda installé…

```
panda install Bailador
```

Vous disposez désormais de tous les outils nécessaires !

2. Vos premières pages grâce aux vues

Nous sommes désormais prêts pour écrire nos premières applications Web.

i

Lorsqu'on crée une application Web, c'est-à-dire un site Web qui évolue en fonction de l'utilisation de ses visiteurs, le code peut rapidement s'allonger. Pour se simplifier la tâche, on aime séparer le code en plusieurs **couches**.
Une partie du logiciel s'occupera d'aller chercher des données là où elles se trouvent (base

de données, API, réseaux sociaux...) et une autre s'occupera de les afficher. On appelle cette dernière couche les **vues**. C'est d'elles que nous allons parler.

Pour débuter, nous allons faire simple : une simple page qui affichera « Hello World ! ». Pour cela, créez un fichier qui contiendra le code de votre application (pour moi, ce sera `app.p6`). Écrivez-y le code suivant et exécutez-le.

```
1 use v6;
2
3 use Bailador;
4
5 get '/' => sub {
6     'Hello World !'
7 }
8
9 baile;
```

```
1 $ perl6 app.p6
```

Comme vous l'indique votre console, vous devez vous rendre sur l'adresse localhost :3000 pour pouvoir accéder à votre application.

Nous allons désormais tenter de comprendre ce qui s'est passé ici. La première ligne de notre programme, `use v6` sert à indiquer que nous utilisons Perl 6, et non pas une version antérieure. La troisième ligne permet d'inclure les fichiers de Bailador, afin de pouvoir en utiliser les différentes fonctions. Nous associons ensuite à une URL une fonction. Grâce à la fonction `baile`, nous lançons notre application.

Comme vous pouvez vous en douter, une application ne se résume pas à une seule et unique URL. Vous pouvez en créer une multitude.

```
1  use v6;
2
3  use Bailador;
4
5  get '/index' => sub {
6      'Bienvenue sur mon site !'
7  }
8
9  get '/forum' => sub {
10     'Bienvenue sur le forum !'
11 }
12
13 get '/contact' => sub {
```

```
14      'Vous êtes sur la page contact.'
15  }
16
17  baile;
```

Listing 1 – app.p6

Jusqu'à présent, nous avons uniquement renvoyé du texte. Mais heureusement, nous pouvons également renvoyer du code HTML à notre navigateur. Vous pouvez tester par vous-même.

```
1  get '/' => sub {
2      '<h1>Bienvenue sur mon site !</h1>'
3  }
```

Nous disposons désormais de plus de possibilités ! Toutefois, vous réaliserez rapidement qu'écrire tout votre code HTML de cette manière devient un vrai cauchemar. C'est pourquoi les *templates* ont été créés. Nous verrons dans la suite de ce tutoriel comment envoyer au client un fichier, qui contiendra notre code HTML.
Vous le savez peut-être déjà : des informations peuvent être transmises via nos URL. Par exemple, si nous souhaitons dire bonjour à notre visiteur, notre URL pourrait ressembler à ça : `/hello/pseudo`. Chaque visiteur pourrait alors se voir afficher un bonjour personnalisé. Voici le code que nous utilisons pour parvenir à cette tâche avec Bailador.

```
1  use v6;
2
3  use Bailador;
4
5  get '/hello/:pseudo' => sub ($pseudo) {
6      "Hello $pseudo !"
7  };
8
9  baile;
```

Vous pouvez tester différentes URL. Par exemple, localhost :3000/hello/Clem . Ci-dessous, voici en bonus un autre cas pratique que vous pourriez être amené à utiliser.

```
1  get '/post/:post_id' => sub ($post_id) {
2      "Post n°$post_id"
3  };
```

Actuellement, ce code possède une faille, qui peut être exploitée par n'importe quelle personne mal intentionnée. Je vous laisse vous référer à la dernière partie pour en savoir plus.

3. Utiliser des templates

Comme nous l'avons vu dans la partie précédente, il n'est pas uniquement possible d'envoyer du texte au navigateur. Nous pouvons également envoyer du code HTML. Une petite piqûre de rappel ne vous fera pas de mal.

```
use v6;

use Bailador;

get '/' => sub {
    '<html>
       <head>
         <title>Mon site Web</title>
       </head>
       <body>
         <h1>Bienvenue !</h1>
         <p>Bienvenue sur mon site Web, créé avec Bailador !</p>
        </body>
    </html>'
}

baile;
```

Listing 2 – app.p6

Vous pouvez de cette façon envoyer du code HTML à votre navigateur. Toutefois, cette méthode n'est absolument pas pratique. C'est pour cette raison précise que les *templates* ont été créés. Ceux-ci vont nous permettre d'envoyer au navigateur un fichier contenant notre code HTML. Les *templates* permettent en quelque sorte de produire du code HTML et de pouvoir y insérer le contenu d'une variable. Pour vous donner un exemple, PHP lui-même est un langage de *templates* : au milieu du code HTML, nous pouvons insérer une variable PHP.

```
<p>Hello <?php $pseudo ?></p>
```

Sauf que dans notre cas, le PHP ne nous intéresse pas. Il existe plusieurs moteurs de *templates* pour Perl6 : Hinges , Web : :Template , Template6 , ou encore Template Mojo . Bien sûr, cette liste n'est pas complète. N'hésitez pas à chercher le moteur qui vous conviendra le mieux. Au cours de ce tutoriel, j'utiliserai Template : :Mojo. Vous pouvez l'installer manuellement, ou plus simplement, à l'aide de panda. Nous disposons désormais des outils nécessaires pour remplir notre tâche : envoyer au navigateur notre code HTML dans un fichier.
Dans ce but, créons un fichier HTML et enregistrons le dans un répertoire nommé `views`. Comme nous utilisons Template : :Mojo, nos *templates* porteront l'extension `.tm`. Pour moi, ce sera `page.tm`.

```
<!doctype html>
<html>
```

```
    <head>
      <meta charset="utf-8" />
      <title>Mon site Web</title>
    </head>

    <body>
      <h1>Bienvenue !</h1>
      <p>Bienvenue sur mon site !</h1>
    </body>
</html>
```

Listing 3 – page.tm

Le dossier contenant votre application doit ainsi ressembler à ceci.

```
app.p6
/views
  page.tm
```

Bailador utilise sa fonction `import` afin de connaître l'emplacement de votre dossier `views`. Actuellement, Rakudo n'appelle pas automatiquement cette fonction. Nous devons donc l'inclure manuellement, comme ceci.

```
use Bailador;
Bailador::import;
```

Nous arrivons au but. Comment faire pour renvoyer un *template* ? C'est tout simple ! Il suffit d'utiliser la fonction `template`, comme ceci.

```
get '/' => sub {
    template 'page.tm'
}
```

Notre navigateur nous affiche désormais notre page HTML, comme nous le souhaitions. C'est quand même beaucoup plus simple ainsi !
Pour les curieux, découvrons un peu plus en profondeur le fonctionnement de la fonction `template`. Que se passe-t-il en réalité lorsque nous faisons appel à cette fonction ? Notre fichier HTML est lu entièrement, stocké, et envoyé au navigateur.
Avec Perl, pour lire un fichier, la fonction `slurp` est utilisée. Il ne nous reste plus qu'à stocker le résultat obtenu dans une variable, qui sera renvoyée au navigateur.

```
get '/' => sub {
    my $html = slurp 'views\page.tm';
    return $html;
}
```

Les deux précédents codes auront un résultat identique.

Nous n'en avons pas fini avec les *templates*. Souvenez-vous, dans la première partie, nous avions vu afficher un bonjour personnalisé en fonction de l'utilisateur. Comment pouvons-nous effectuer cette tâche à l'aide des *templates* ? C'est ce que nous allons découvrir.
Les *templates* nous permettent d'inclure du code Perl dans nos pages HTML, qui sera exécuté avant d'être renvoyé à votre client. Les lignes débutant par le signe `%` seront considérées comme du code Perl6. Si vous souhaitez inclure du code dans des balises HTML, il vous faudra utiliser les symboles suivants : `<%` et `%>`.
Nous pouvons par exemple être amenés à devoir répéter dix fois la même phrase. Pas de soucis !

```
% for 0..9 {
    <p>Qui sème le zeste récolte le savoir.</p>
% }
```

Et si nous souhaitons donner un bonjour personnalisé à notre utilisateur, nous pouvons insérer une variable dans nos balises HTML.

```
% my $pseudo = 'Clem';
<p>Bonjour <% $pseudo %> !</p>
```

4. Les requêtes

Il existe deux principaux types de requêtes, que vous connaissez certainement déjà : les requêtes POST et les requêtes GET. Les requêtes GET sont majoritairement utilisées pour demander une ressource, telle qu'une page Web. Les requêtes POST permettent, elles, de modifier une ressource. Les données transmises sont envoyées dans le corps de la requête.
Dès la première partie, nous avons utilisé les requêtes GET, en transmettant des informations via nos URL. Rappelez-vous-les…

```
get '/post/:post_id' => sub ($post_id) {
    "Post n°$post_id";
}
```

Comme dit plus haut, ce petit bout de code présente une faille de sécurité. En informatique, il ne faut **jamais** faire confiance à l'utilisateur. En temps normal, vous vous attendez à ce que l'utilisateur entre un chiffre, correspondant à l'identifiant de l'article. Toutefois, rien ne vous protège de tomber sur un utilisateur mal intentionné, ou tout simplement maladroit : que se passera-t-il si, au lieu de fournir un nombre, il fournit une chaîne de caractères ?
Pour pallier ce problème, il est primordial de **toujours** vérifier les données entrées par l'utilisateur. Nous pouvons procéder de la sorte.

```
use v6;

use Bailador;

```

```
get '/post/:post_id' => sub ($post_id) {
  try {
    $post_id.Int();
    return "Post n°$post_id";
  }
  return 'Identifiant invalide !';
};

baile;
```

Nous pouvons également imaginer un cas dans lequel notre URL devrait contenir plusieurs paramètres, comme l'identifiant du post, ainsi que son titre. Cette opération peut s'effectuer tout simplement ainsi.

```
get '/post/:post_id/:titre' => sub ($post_id, $titre) {
  try {
    $post_id.Int();
    return "Post n°$post_id : $titre";
  }
  return 'Identifiant invalide !';
};
```

Intéressons nous désormais aux requêtes POST. Nous ne transmettons plus nos informations via nos URL, mais via le corps de la requête. Une des applications les plus courantes se trouve dans les formulaires. Prenons cet exemple…

```
<form method='POST' action='/hello' />
    <input type='text' name='pseudo' placeholder='pseudo' />
    <input type='submit' />
</form>
```

Ici, le pseudo choisi par l'utilisateur sera transmis via une requête POST, comme indiqué. Voyons comment gérer cette situation côté serveur.

```
post '/hello' => sub {
    my $pseudo = request.params<pseudo> // '';
}
```

Nous stockons ici la valeur envoyée par l'utilisateur dans une variable également nommée `$pseudo`. Il ne nous reste plus qu'à dire bonjour à notre utilisateur.

```
post '/hello' => sub {
    my $pseudo = request.params<pseudo> // '';
    return "Hello $pseudo !";
}
```

Si vous avez un quelconque doute, voici le code complet de notre petite application.

```
# app.p6
use v6;

use Bailador;

get '/' => sub {
    return "<form method='POST' action='/hello' />
                <input type='text' name='pseudo' placeholder='pseudo' />
                <input type='submit' />
            </form>"
};

post '/hello' => sub {
    my $pseudo = request.params<pseudo> // '';
    return "Hello $pseudo !";
};

baile;
```

Ce premier tutoriel touche à sa fin. J'espère que vous avez apprécié cette découverte de Bailador, et que vous avez désormais envie d'en découvrir plus. N'hésitez pas à créer vos propres applications, la pratique reste le meilleur moyen d'apprendre !

J'en profite pour remercier :

— nohar , rédacteur originel de l'introduction à Perl ;
— Dominus Carnufex pour ses corrections ;
— artragis , pour la validation de ce tutoriel.