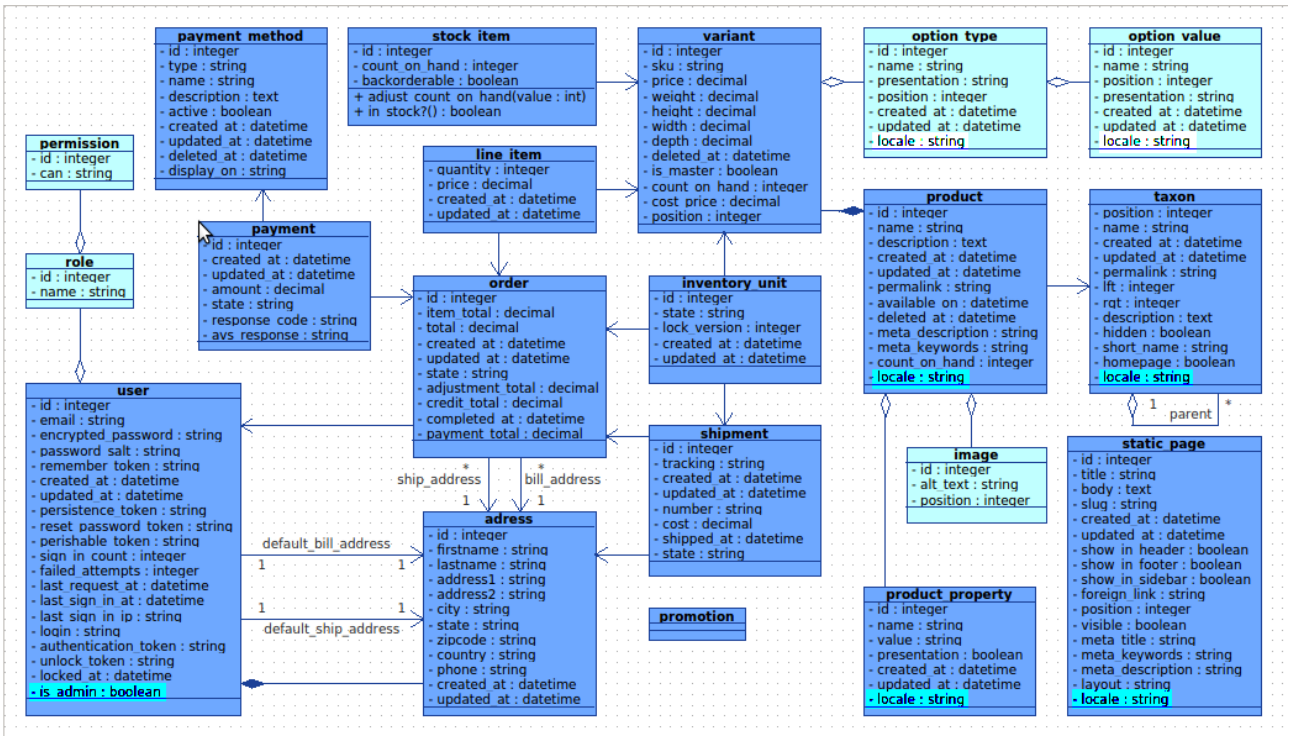
Compte rendu de réalisation

Table of Contents

Changements effectués au niveau conceptuel

La partie d'analyse et de conceptualisation de l'application nous a pris beaucoup de temps au démarrage du projet. Par conséquent, les modifications apportées au schéma relationel et à l'architecture de l'application sont vraiment mineures.
Afin de réaliser l'internationalisation de l'application, nous avons du introduire la notion de localisation sur certains modèles de données. Les utilisateurs sont à présent hiérachisés et possède un éventuail de permissions regroupé dans un ou plusieurs roles. Les autres changements sont pour la plupart de simples oublis.

Changements sur le diagramme des classes



Nous avons du au cours du projet apporter certaines modifications au schéma relationnel. Ces changements n'ont pas été très difficiles à mettre en œuvre grâce à notre utilitaire de migration développé en interne. Les changements ont été les suivants :

- L'entité « permission » : En plus d'une séparation store/backoffice, le panel d'administration comprend de nombreuses fonctionalités succeptibles d'invoquer différents acteur. Pour chacunes des actions possibles (ex : créer un produit, bannir un compte utilisateur, internationaliser une page statique), il nous sera possible de définir un hook sur l'opération permissible, vérifiant l'éligibilité de l'utilisateur avant de l'effectuer, exemple :
```
if $current_user_roles->permissions->find({ can => 'update_product' })
      $self->product->update( $self->params('product') );
else
      $self->flash( error => l('unsufficient_privileges') );
return $self->redirect_to('admin_products');
```
- L'entité « role » : Permet de définir un ensemble de permissions.
- L'attribut « user.is_admin » : Cet attribut est uniquement à des fin d'optimisation. En effet, l'accès au backoffice ne n'est pas défini par un rôle mais par ce booléen.
- Les entités « option_type » et « option_value » : Cette table permet de définir des spécifications aux variantes de produits qui peuvent avoir une influence sur le prix (contrairement au « product_proprietes »). Par example, le prix d'un T-Shirt sera plus cher s'il possède une option de type « taille » et de value « XL »

- L'entité « image » : Simple oubli, images associés au produit. L'attribut « position » permet de determiner l'ordre dans la présentation.
- L'attribut « locale » : Utilisé pour l'internationalisation de la base. Il sera toujours couplé à l'id pour définir la clé primaire. De cette maniere, pour un meme produit, « locale » sera la référence de la langue pour le tuple , par exemple :
    → Product id=123, locale='fr-fr', name='Machine à laver', ...
    → Product id=123, locale='en-us', name='Washing Machine', …

Changements sur l'architecture de l'application

▸ db/	Templates SQL de creation, migration et population de la base
▸ doc/	Toute la documentation du projet
▾ lib/	Modules perl relatifs au projet
▾ PerlEcommerce/	Module de base pour l'application PerlEcommerce
▸ Command/	Utilitaires CGI pour les scripts à utiliser en ligne de commande
▸ Controller/	Définitions de tous les controllers de l'application
▸ I18N/	Internationnalisation de toutes le données statiques
~~▸ Model/~~	Définition des modèles de donnée (génération auto depuis DBIx)
~~▸ Result/~~	Requêtes SQL pour chacun des modèles (supprimé depuis DBIx)
▸ Schema/	Schema relationnel-objet complet de l'application
PerlEcommerce.pm	Module Racine
▾ public/	Répertoire pour les fichiers statiques
▸ css/	Feuilles de styles de l'application
▸ font/	Polices utilisées par l'application
▸ img/	Images de l'application et des images uploadées
▸ js/	Fichers Javascipt pour l'application client
▸ perldoc/	Documentation statique du module PerlEcommerce
▾ script/	Scripts d'aide à la mise en place de l'application
bundle	Permet d'installer toutes les dépendances
migrate	Migration de la base de donnée + Génération des modèles
seed	Population de la base avec données exemples
server	Démarrage du server
test	Lancement des tests unitaires
▸ t/	Definition des tests unitaires
▸ templates/	Tous les templates de l'application, traités par TTRenderer
perl-ecommerce.conf	Configuration de l'application (base de donnée, smtp, upload limit, ...)
README.md	Document de présentation du projet
.gitignore	Fichiers à ignorer pour la synchronisation avec le dépôt distant

Choix Techniques

Librairies

Mojolicious

Retour dans les premiers jours de l'Internet, beaucoup de gens ont appris Perl en raison d'une magnifique bibliothèque Perl appelé CGI. C'était assez simple pour commencer sans trop savoir la langue et assez puissant pour vous tenir en haleine, l'apprentissage par la pratique a été très amusant. Alors que la plupart des techniques utilisées sont dépassées maintenant, l'idée sous-jacente ne l'est pas. Mojolicious est une nouvelle tentative de mise en œuvre de cette idée en utilisant l'état de la technologie de pointe.

Caractéristiques

- Server web en temps réel, permettant de développer facilement des applications à fichier unique grâces aux prototypes bien structurés de Mojolicious :: Lite
  - Puissant avec des parcours RESTful, des plugins, des commandes, des modèles Perl-ish, la négociation de contenu, la gestion de session, les test unitaires, serveur de fichiers statiques et le premier support de la classe Unicode.
- Très propre, orientée API pure Perl portable et objets sans aucune magie cachée et aucune exigence en dehors de Perl 5.10.1 (5,16 + recommandée, et les modules CPAN optionnelles seront utilisés pour fournir des fonctionnalités avancées s'ils sont installés).
- Pleine pile HTTP et WebSocket client/serveur mise en œuvre avec les protocole IPv6, TLS, SNI, IDNA, Comet (long polling), keep-alive, multipart, timeout, cookie, multipart, proxy et le support de la compression gzip.
- Serveur web d'E/S antibloquant, supportant de multiples boucles événement, le preforking facultatif et le déploiement à chaud, parfait pour l'intégration de conteneurs
- CGI automatique et détection PSGI.
- Parseur JSON et HTML / XML intégré avec support pour les sélecteurs CSS.
- Philosophie de programmation fraîche issue d'années d'expérience dans le développement Catalyst. (Framework MVC pour perl et apache très populaire)

Catalyst est en effet un framework très puissant et réputé chez les Perl Monks. La communauté semble cepandant le délaisser pour des solutions plus légères et une certaine nouveau conptuelle. L'application est largement inpirée de Ruby-on-Rails, un mamouth dans sa catégorie pour le développement d'applications à architecture MVC qui intègre un eventail d'outils impressionant (routes, hooks, helpers, tests unitaires, ..) et une inuitivité insolente.

Malgré toutes ces fonctionnalités, Mojolicious n'intègre malheureusement pas de support de connexion à une base de donnée, ou un équivalent à Active::Record (RoR). C'est pourquoi nous avons été amenés à implémenter nos propres modeles de données.

DBIx::Simple

DBIx::Simple fournit une interface simplifiée pour DBI, le module de base de données Perl.

Ce module vise à un développement rapide et un entretien facile. Préparation et exécution des requêtes sont combinées en une seule méthode, l'objet de résultat (qui est un wrapper autour de la déclaration) fournit des méthodes ordonées par rangée et siphonage faciles.
Cette librairie présente un invénient majeur, chacune des requêtes doivent êtres écrites à la main et

le binding des variables devient vite un casse tête lorsque l'aspect relationnel des modèles se corse. Nous avons vite réalisé, vu l'ampleur de notre schéma relationnel, que ce choix technologique nous ferait perdre un temps précieux dans l'implémentation de nos classes.

DBIx::Class::Schema::Loader

DBIx est un ensemble de modules Perl permettant de triturer toute base de données sans en connaitre précisement le language. La base de ces modules s'appuie sur **DBIx::Class**, ce dernier fourni l'abstraction aux base de données et représente les tables comme de simples classes. L'accès aux données devenant alors trivial.
Pour comprendre DBIx il faut en connaitre le vocabulaire, qui se résume à:
ResultSource : Il s'agit ni plus ni moins de la table elle meme.
ResultSet: C'est la requete
Nous avons la table et la requête, il nous manque encore le plus important: la base de donnée et bien sûr la défnition des tables de cette dernière: c'est **le schema**. Là ce sera le role que **DBIx::Class::Schema** devra assurer.
Le choix spontané de cette librairie à débouché sur un sprint d'une semaine pour assurer le migration complète du schéma. Semaine durant laquelle l'utilitaire de migration devra être achevé : Script de migration et schema SQL de la base de donnée + Script de génération des modèles PerlEcommerce::Model
Une fois le sprint terminé, les tâches ont pu être plus facilement distribuées puisque l'implémentation des vues et controleurs implique peu de conflits
Pour migrer la base et générer tous modeles, il suffit maintenant de rentrer la commande :

```
$ perl script/migrate
```

Et pour insérer les données d'initialisation (ex : création d'un super utilisateur) et le catalogue de test dans la base :

```
$ perl script/seed
```

Image::Magick

Comme toute production pour le web 2.0 qui se respecte, l'optimisation en termes de présentation est primordiale. Il nous sera donc nécéssaire de générer un échantillon d'images de différentes tailles et ratio selon le contexte d'affichage d'un produit, par example :

- « thumb.png» : `100 x 100 px`
  Miniature, utilisée pour l'affichage d'un produit d'une liste. L'image est ici recadrée.
- « thumb@2X.png » : `200 x 200 px`
  Miniature pour écran rétina (IPhone, MacBook, …).
  Le résolution de l'image doit être doublée.
- « large.png » : `500 x <auto> px`
  Image pour la présentation produit, la hauteur est ajustable puisque la présentation est en colonnes et la hauteur du conteneur des détails du produit n'a pas de hauteur fixe non plus.

ImageMagick est une suite de logiciels pour créer, éditer, composer, ou convertir des images bitmap. Il peut lire et écrire des images dans une variété de formats (plus de 100) dont DPX, EXR, GIF, JPEG, JPEG-2000, PDF, PhotoCD, PNG, PostScript, SVG et TIFF. Il permet de redimensionner, pivoter, miroir, rotation, distorsion, de cisaillement et de transformer des images, ajuster les couleurs de l'image, appliquer divers effets spéciaux, ou dessiner du texte, des lignes, des polygones, des ellipses et des courbes de Bézier (yahoo!).

La fonctionnalité de ImageMagick est généralement utilisé à partir de la ligne de commande mais nous pouvons égalemement utiliser ces fonctionnalités à partir de programmes écrits dans notre langage préféré : G2F (Ada), MagickCore (C), MagickWand (C), ChMagick (Ch), ImageMagickObject (COM +), Magick + + (C + +), JMagick (Java), L-Magick (Lisp), Lua,

NMagick (Neko / haXe), MagickNet (. NET), PascalMagick (Pascal), **PerlMagick (Perl)**, MagickWand pour PHP (PHP), imagick (PHP), PythonMagick (Python), RMagick (Ruby), ou tclmagick (Tcl / Tk).

ImageMagick est un logiciel libre livré sous forme de distribution de prêt-à-l'emploi binaire ou sous forme de code source que nous pouvons utiliser librement, copier, modifier et distribuer des applications à la fois ouverts et propriétaires. Il est distribué sous la licence Apache 2.0. Le processus de développement ImageMagick assure une API stable et perenne.

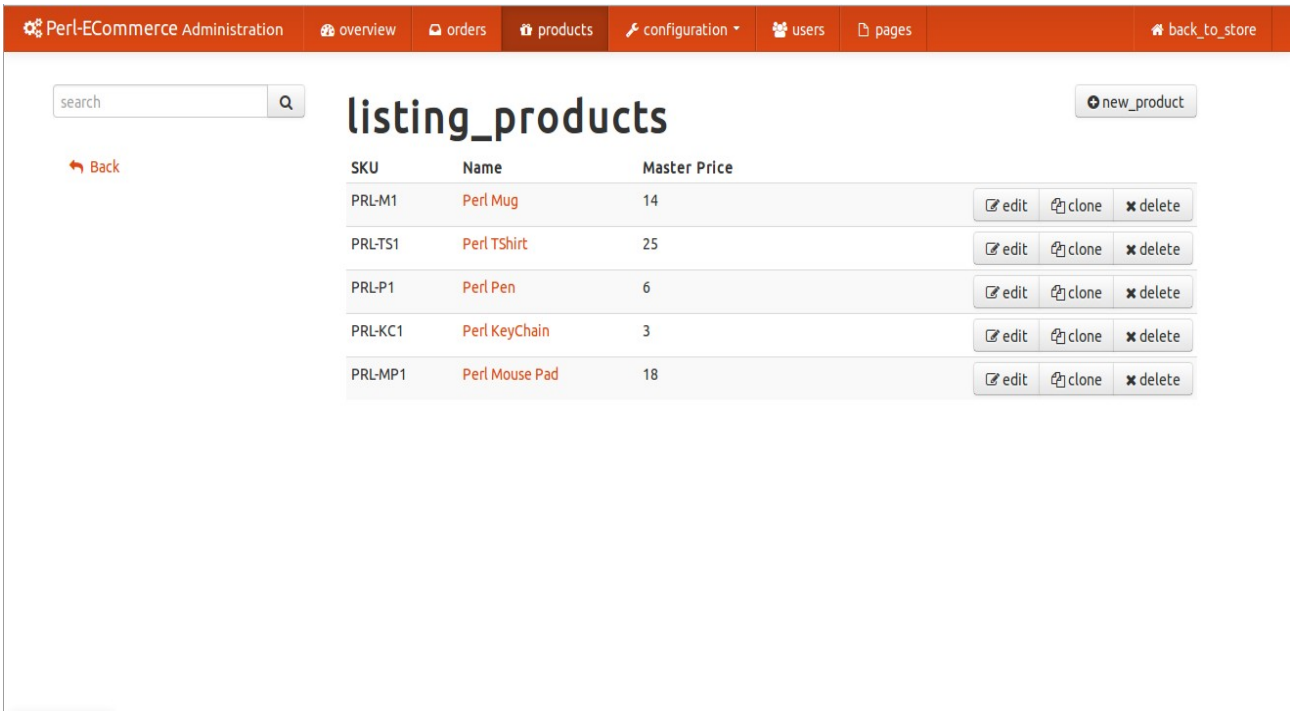
jQuery

jQuery est une bibliothèque JavaScript rapide, légère et riche en fonctionnalités. Cela rend des choses comme le parcours et la manipulation d'un document HTML, la gestion des événements, l'animation et les requêtes Ajax beaucoup plus simple avec une API facile à utiliser qui fonctionne sur une multitude de navigateurs. Avec une combinaison de polyvalence et d'extensibilité, jQuery a changé la façon dont des millions de gens écrivent JavaScript.

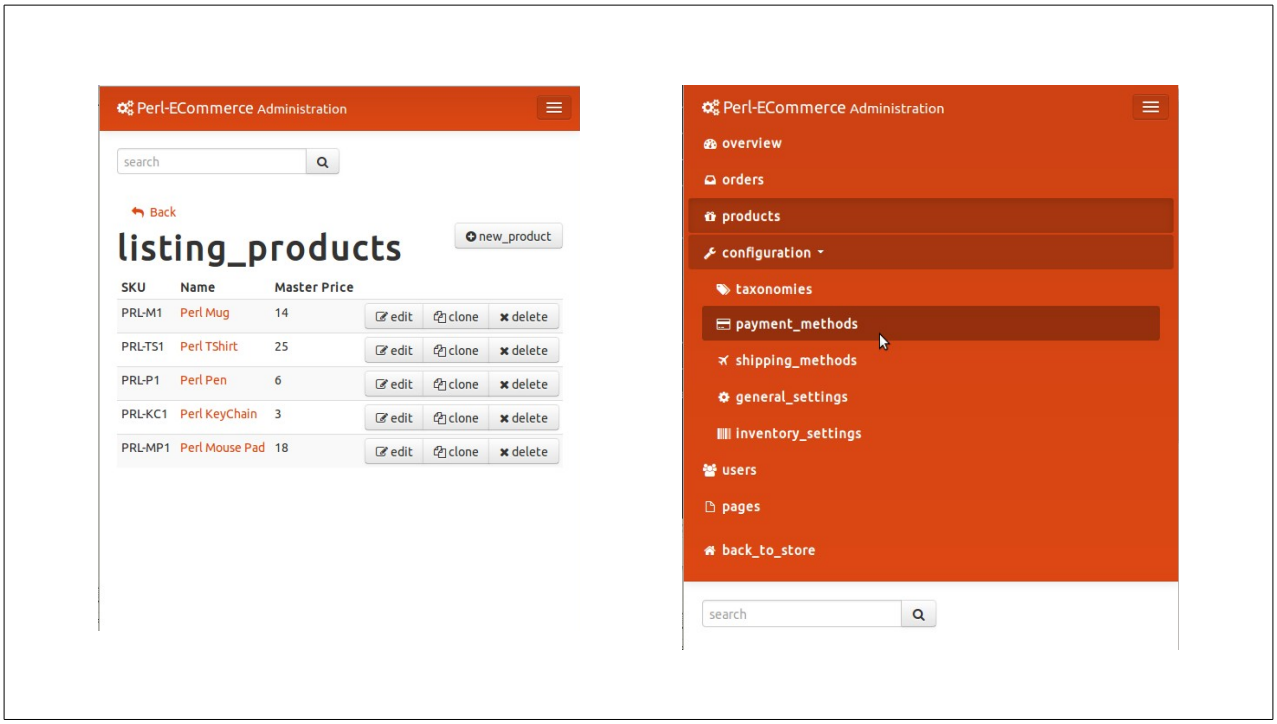
Twitter Bootstrap

Twitter Bootstrap est une collection d'outils utile à la création de sites et applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option. C'est l'un des projets les plus populaires sur la plate-forme de gestion de développement GitHub, et ce n'est pas pour rien. Cette outil nous a permis la mise en œuvre en un temps record d'une interface non seulement compatible cross-navigateur mais aussi cross-plateforme. Le site est entierment comptible Tablette, Mobile, Notebook etc.

Listing des produits sur un ordinateur de bureau :



Listing des produit sur un mobile



Tests unitaires

Comme le standard d'application Perl l'impose, nos tests unitaires sont situés dans le répertoire « t/ » du projet. Ils pemettent de tester toutes les fonctionnalités de l'application une par une afin d'effectuer une validation utlime du système avant la sortie d'une version.
Pour mener à bien ces tests, nous avons utilisé les librairie Test::More et Test::Mojo qui nous permette de valider chacune des routes HTTP du système et d'en évaluer les responses.

Exemple :

```
  use Test::More;
  use Test::Mojo;

  my $t = Test::Mojo->new('PerlEcommerce');

  # Affichage de la page d'accueil et vérification du texte
  $t->get_ok('/')->status_is(200)->text_is('div#welcome' => 'Welcome !');

  # Ajout d'un produit avec ajax et verification d'un nœud dans le json
  $t->post_ok('/products.json' => form => {q => 'Perl'})
    ->status_is(200)
    ->header_is('Server' => 'Mojolicious (Perl)')
    ->header_isnt('X-Bender' => 'Bite my shiny metal ass!')
    ->json_is('/results/4/name' => 'Perl T-Shirt');

  done_testing();
```

Afin de réaliser tous les tests unitaires de l'application, nous avons écrit un script capable de fournir un rapport complet quant à l'évaluation de chacune des opérations possibles :

```
    $ perl script/test
```

Documentation du code

Pour la documentation, puisque Sphinx n'est pas (encore) disponible pour Perl, nous avons décidé de suivre la documenttation standard « perldoc ». Il nous est alors possible de decrire les spécificités de chacun des modules perl implémenté dans le meme fichier que ce dernier. Malgré les redondances de codes qu'implique perldoc en n'utilisant pas le principe des décorateurs, le rendu est assez sympa.
La documentation pdf du module PerlEcommerce est disponible dans le répertoire « doc/perldoc » du projet. Il est également possible d'accéder à un site statique de documentation intégré à l'application disponible à l'addresse http://localhost:3000/perldoc/PerlEcommerce.html
et bientôt sur http://search.cpan.org/

Manuel utilisateur

Un manuel utilisateur est disponible dans le répertoire de documentation du projet
« doc /manuel_utilisateur.pdf »

Qualité logicielle

Tout au lont du projet : est ce que le plan qualité a bien été respecté ? Est-ce qu'il etait top contraignant, trop directif.

Conclusion et difficultés rencontrés

(techniques et orgnisationelle)