Développement d'applications web avec Ocsigen

Xavier Van de Woestyne

Decembre 2014

Pendant très longtemps, j'ai été amené à concevoir des applications web. J'ai eu l'occasion d'expérimenter plusieurs technologies (PHP, Ruby on Rails, Django, Spring, ASP.NET et Yaws + Erlang). Je dois avouer n'avoir été que trop rarement satisfait par ces outils, principalement car depuis ma découverte avec les langages fonctionnels statiquement typés, j'ai réellement du mal à m'en passer. C'est pour cette raison qu'à l'annonce de Ocsigen, j'ai été véritablement emballé. Cependant, Ocsigen est, actuellement, pauvre en ressource et assez dépaysant (cette opinion n'est absolument pas à prendre comme un reproche mais plus comme une constatation et s'explique par l'innovation démontrée dans Ocsigen et sa jeunesse)), j'ai donc décidé de lui accorder un petit article proposant à sa fin, une implémentation pratique où chaque étape sera décrite !

Présentation sommaire de Ocsigen

Ocsigen est une suite de logiciels libres écrite en OCaml, développée par le laboratoire français PPS. Son objectif principal réside dans la conception d'applications web fiables au moyen du langage de programmation OCaml qui est, en plus d'être extrêmement fiable, très expressif.
Ocsigen propose des réponses à des problèmatiques récurrentes du développement web, dans certains cas assez différentes des solutions proposées par d'autres technologies plus populaires.

Un des arguments en faveur d'Ocsigen est l'usage *d'un seul langage pour tout faire*, en effet, qu'il s'agisse de l'écriture de la couche statique (le HTML, vérifiant la validité de ce dernier), du Javascript ou du SQL (vérifiant la bonne formation d'une requête), Ocsigen est dôté d'outils et d'extensions de syntaxes pour ne demander au programmeur, qu'une connaissance de Ocaml.

Bien qu'en règle général, j'aime ne pas m'enfermer dans une forte dépendance à un outil, je dois avouer que l'excès de flexibilité de Javascript m'a quelque fois écoeuré. En écrivant du Javascript avec

> OCaml, on préserve un des intérêts majeur du langage (de mon point de vue), son typage fort. Le déboguage d'applications usant massivement de Javascript n'est plus un problème.

N'utilisant qu'un seul langage, Ocsigen permet de manipuler les couches *client-serveur* de manière aisée, de la même manière que Meteor.js. Il parait que l'on appelle ça *une plate-forme de développement isomorphique*, mais je ne suis pas convaincu de l'apellation.

L'organisation des fichiers et des points d'entrés d'une application réalisée avec Ocsigen est aussi très différente de ce que l'on peut habituellement voir. En effet, toute la navigation est articulée autour d'une notion de service, correspondant à une valeur OCaml à part entière et pouvant être interconnecté à d'autre services. Un service peut être caractérisé par un chemin d'accès, et des paramètres **GET** et **POST** statiquement typés. (De même qu'un service ne peut être qu'une action atomique, sans chemin particulier, nous parlerons dans ce cas de coservice). Les applications Ocsigen étant compilées, un lien ne peut pointer vers un service inexistant, une vérification complémentaire des liens cassés est donc effectué à la compilation.

A ça s'ajoute *Lwt*, une bibliothèque pour effectuer des traitements concurrent, soit une fragmentation de tâches en sous-programmes s'exécutant simultanéments (où chaque processus décide lui même de céder la main à son successeur). Cette bibliothèque apporte un confort indéniable dans la construction de tâches asyncrones et son portage pour **Js_of_OCaml** (le Javascript statiquement typé de Ocsigen) est admirablement pertinent dans le cadre d'exécution de Javascript.

A ces axes brievement présenté s'ajoute une collection d'outil pour résoudre des problèmatiques classiques du développement web, que nous tâcherons de survoler dans la partie pratique de cet article.

Pourquoi utiliser Ocsigen

Si j'ai décidé de m'intéresser à Ocsigen, c'est avant tout par intérêt pour le langage OCaml. Cependant, après usage, il est évident que, comme pour le développement d'application classique, le typage statique est véritablement un atout indéniable. On évite une quantité de tests chronophage, et beaucoup de vérifications "plus bas niveau" sont effectuées par Ocsigen. Le développeur n'a donc plus à s'en soucier. De plus, le fait, par exemple, que le compilateur vérifie la bonne sémantique du HTML accélère le flot de travail (n'obligeant plus à vérifier chaque fois la validité de son code HTML).

De mon point de vue, Ocsigen se pose de bonnes questions face à des problèmatiques récurrente. Et même si je ne pense pas qu'il devienne (même si je l'aimerais) un incontournable absolu, au même titre que PHP ou Ruby On

Rails, je pense qu'il met en lumière certaines bonnes pratiques, qui seront sûrement adoptées par d'autres outils, potentiellement plus démocratiques. (On peut noter une certaines similitude avec le projet OPA, par exemple).

Des utilisateurs d'Ocsigen

Ocsigen n'est actuellement pas un des mastodonte de la programmation web, cependant, il possède tout de même une liste d'utilisateurs. En voici quelques'uns choisis à la volée :

- Facebook : Utilisation de Js_of_OCaml (pour leur développement interne).
- Cumulus : Un petit outil de partage de liens (dont le code source m'aura été très utile pour mon apprentissage).
- BeSport : Ils viendraient de recruter Vincent Balat, le chef du projet Ocsigen.
- Prumgrana : Un projet qui semble gagner tous les concours où il s'inscrit.

Quoi qu'il en soit, je n'ai pas eu l'occasion de lire de témoignage désaprobateur sur Ocsigen, sauf peut être … le mien, après avoir tenté un Hackathon sans une préparation suffisante… mais sachez que j'ai revu mon opinion sur le projet après avoir pris le temps de me plonger dedans.

Implémentation d'une plateforme de micro-blogging

Pour présenter l'implémentation concrète d'une application avec Ocsigen, j'ai choisi un exemple bien peu original. Cependant, je pense qu'il est intéressant car il permet de présenter plusieurs aspects du framework (création d'un **CRUD**, donc usage d'une base de données et de Macaque, utilisation de **ocaml-safepass** pour crypter les mots de passes et éventuellement l'exposition et l'usage de **webservices**).

> Avant de me lancer dans l'explication détaillée, je tiens à préciser que cet article est aussi une manière d'éprouver Ocsigen, pour moi, dans un contexte très pratique. Je ne suis pas du tout un expert et il est possible que certains choix structurels soient discutables. Si vous avez des choses à redire n'hésitez pas à vous servir des commentaires, je suis ouvert à toute critique !

Par soucis de lisibilité (et car ça ne présente pas beaucoup d'intérêt), le CSS sera mis de côté. On évoquera comment utiliser sa propre feuille de style, mais je ne parlerai pas du code CSS quand il n'aura pas de rapport direct avec l'implémentation de l'application.

Installation

L'installation d'Ocsigen est véritablement simple depuis la mise en place de OPAM. Après l'avoir installé (et initialisé), l'installation de Ocsigen peut se limiter à :

```
opam install eliom
```

Pour ma part, j'utilise la version `4.01.0` de OCaml et l'installation (un peu longue) s'est déroulée sans soucis. L'usage d'OPAM est un véritable plus car c'est lui qui nous permettra d'installer les paquets additionnels nécéssaire à notre application. (On pourrait faire une analogie avec les *gems* de Ruby par exemple).
Sans rentrer dans les détails de l'anatomie du paquet, voici quelques petits compléments explicatifs sur la constitution du paquet `eliom` :

- `Js_of_ocaml` : Pour produire du Javascript bien typé (et en ocaml)
- `Macaque` et `PgOcaml` : Pour la construction de requêtes en OCaml
- `Deriving` : Permet de dériver un type dans une représentation (en JSon par exemple)
- `Lwt` : Bibliothèque pour les traitements concurrents
- `` `Camlp4 `` : Ocsigen offre une collection d'extension de syntaxes
- `` `TyXML `` : Une bibliothèque de génération de XML statiquement typé

Le paquet est aussi évidemment composé d'un serveur web, d'outils pour la gestion des calendriers/dates et de beaucoup d'autres outils. Comme vous pouvez le voir, Ocsigen est riche en composants.

Raisonnement général de l'application

Avant de se lancer dans l'écriture de code OCaml, nous allons penser conceptuellement notre application, proposant des diagrammes minimaliste (et la structure de la base de données).

Un utilisateur non connecté peut :

- Lire les messages publiés par les titulaires d'un compte
- Se connecter
- S'enregistrer

Un utilisateur connecté peut :

- Lire les messages publiés par les titulaires d'un compte
- Se déconnecter

- Créer un message
- Afficher ses messages
- Modifier ses messages
- Modifier les paramètres de son compte

Les constituants de cette application sont donc les **messages** et les **utilisateurs**. Un message sera caractérisé par un **identifiant unique**, une **date de publication**, un **utilisateur posteur** et un **contenu**. Alors qu'un utilisateur sera lui caractérisé par un **identifiant unique**, un **pseudonyme**, une **adresse courriel** et un **mot de passe**.

Voici un diagramme réellement minimaliste de notre base de de données. Comme il l'indique, la relation est effectuée au niveau de l'utilisateur, qui permet de relier les messages publiés par un utlisateur. (Rien de bien compliqué).

users	← messages
`user_id` **PK**	`message_id` **PK**
`nickname`	`publication_date`
`email`	`user_id` **FK**
`password`	`content`

Et le code SQL (j'utilise **Postgres**) pour générer la base de données est ici, il n'est pas expliqué de manière méticuleuse car je suppose qu'il ne présente absolument rien de nouveau. Cependant, comme pour la structure choisie pour ce tutoriel, je suis ouvert à toute critique dans les commentaires !

```sql
-- Création des séquences pour l'auto increment
CREATE SEQUENCE users_id_seq;
CREATE SEQUENCE messages_id_seq;

-- Création de la table d'utilisateurs
CREATE TABLE users (
  user_id integer PRIMARY KEY,
  nickname text NOT NULL,
  email text NOT NULL,
  password text NOT NULL
);

-- Création de la table de messages
CREATE TABLE messages (
  message_id integer PRIMARY KEY,
  publication_date timestamp NOT NULL,
```

```
  user_id integer REFERENCES users (user_id)
      ON DELETE CASCADE,
  content text NOT NULL
);
```

Rien de bien compliqué, mais c'est amplement suffisant pour notre exemple. Comme on peut le voir, la table message est reliée par le champs `user_id` et la suppression d'un utilisateur supprimera tous ses messages.