Université de Bordeaux

Architecture Logicielle Distribuée et Adaptative

Freya Web Application

Auteurs :
Elyas Ben Hadj Yahia
Rémi Delassus
David Frenandez
Ryan Herbert

Enseignant :
David Bromberg

27 janvier 2014

Table des matières

1 | Présentation

Freya est une application web conçue pour faciliter la gestion des musées aux conservateurs.
Ce projet a été mis en place grâce à de nombreux outils et frameworks, notamment Maven [1] et Google App Engine [2].
Le dépôt de code source du projet Freya se situe à cette adresse [3].

1.1 Maven

Maven est un outil de gestion de projet. Il permet notamment de spécifier comment un logiciel doit être compilé, et décrire ses dépendances. Il permet aussi d'intégrer des plugins dans des phases de compilation spécifiques, ce qui permet d'avoir un environnement configurable.

1.2 Google App Engine

Google App Engine (GAE) est une PaaS (Platform as a Service) qui fournit un ensemble d'outils afin de faciliter le développement d'applications web. Cette plateforme met à disposition du développeur un runtime Java, une base de données NoSQL (appellée *datastore*), et l'hébergement de l'application web sur les serveurs Google.

Afin d'intéragir avec le datastore, on peut utiliser plusieurs méthodes différentes, comme par exemple l'API Datastore de bas niveau, JPA, JDO, ou encore d'autres bibliothèques tierces comme Objectify. Freya a été conçu en utilisant l'implémentation JPA 2.0 de DataNucleus [4] pour sa compatibilité avec GAE.

GAE fournit aussi une console d'administration très complète pour monitorer l'état de l'application web, le traffic, les requêtes, les logs, etc...

1. Maven : http://maven.apache.org/
2. Google App Engine : https://developers.google.com/appengine/
3. freya : https://github.com/elyas-bhy/freya
4. DataNucleus : https://code.google.com/p/datanucleus-appengine/

2 | Architecture du backend

2.1 Entités

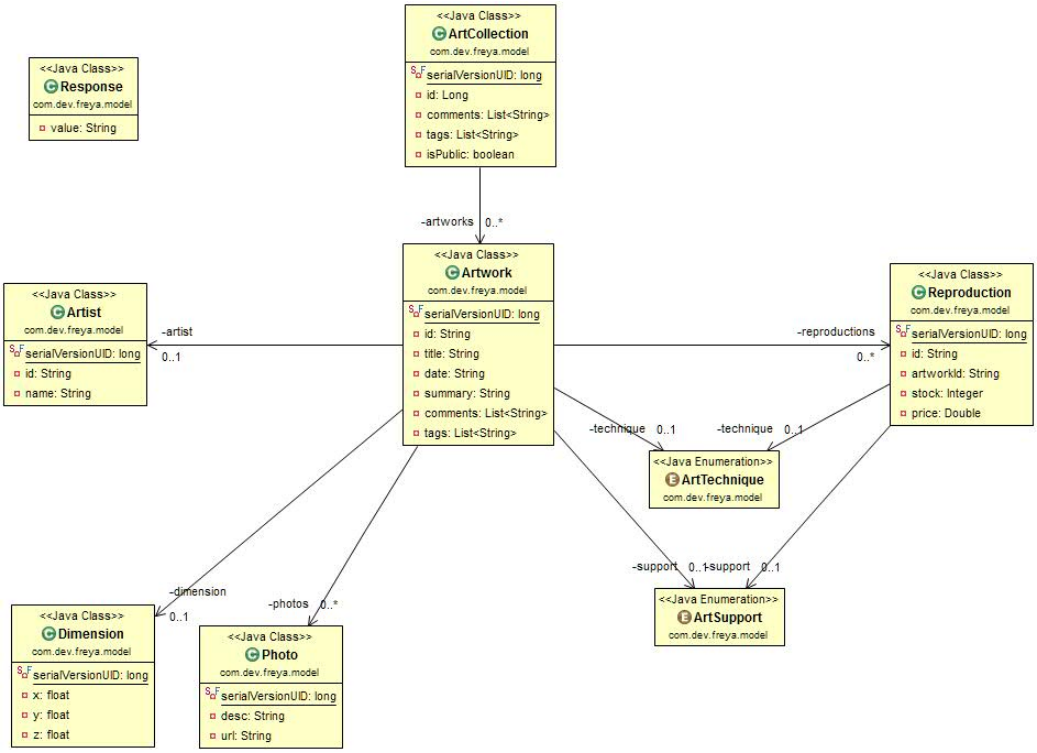

FIGURE 2.1 – Diagramme de classe des entités

2.2 Conception

Une des difficultés majeures de ce projet a été de concevoir correctement les relations entre les entités afin d'être en accord avec les restrictions de GAE. Étant donné que le *datastore* est une base de données NoSQL, il fallait respecter plusieurs contraintes supplémentaires.

Une des contraintes est qu'il faut qu'une entité fille encode la clé de son parent dans sa propre clé. Cela signifie que quand une entité fille est persistée, son parent ne peut plus changer (c.à.d être remplacé par un autre parent). Ceci est dû au fait que les entités fille et parent sont placées dans le même *entity group*, et qu'une entité ne peut appartenir qu'à un seul *entity group*.

Pour des raisons de performance, le *datastore* met en place d'autres restrictions au niveau des requêtes. En effet, certaines opérations ne sont pas permises (subqueries, fonction size, joins complexes, opérateur MEMBER OF sur une collection non-primitive, et d'autres).

Pour gérer les paramètres optionnels pour filtrer les recherches, nous avons utilisé l'API Criteria [1] de JPA afin d'avoir un code propre et maintenable, et qui est beaucoup plus robuste comparé à la création de la requête par concaténation de chaînes.

Afin d'améliorer les performances du backend, nous avons aussi mis en place Memcache [2], un système de cache afin d'accélérer les requêtes d'écriture.

1. JPA Criteria API : http://www.objectdb.com/java/jpa/query/criteria
2. Memcache : https://developers.google.com/appengine/docs/java/memcache/

3 | API REST

L'un des avantages de GAE est qu'il permet de facilement mettre en place une API REST et de l'exposer, grâce au module Google Cloud Endpoints (GCE) [1].

Pour ce faire, on a simplement à annoter une classe comme étant une API, et déclarer les méthodes d'API (notamment routes, paramètres, méthodes HTTP).

Code 3.1 – Exemple d'API avec Google Cloud Endpoints

```
@Api(name = "myapi", version = "v1")
public class MyEndpoints {

    @ApiMethod(
        name = "elements.get",
        path = "elements/{element_id}",
        httpMethod = HttpMethod.GET
    )
    public Element getElement(@Named("element_id") String id) {
        // ...
    }
}
```

GCE se chargera ensuite de construire et déployer l'API avec la description des méthodes exposées.
La description de l'API Freya (v1) se situe à cette adresse [2].
La version déployée de cette API se situe à cette adresse [3].

Exemple d'interaction avec l'API en utilisant curl :

```
curl --header ‘‘Content-Type: application/json’’ -X GET
https://freya-app.appspot.com/_ah/api/freya/v1/artworks
```

1. GCE : https://developers.google.com/appengine/docs/java/endpoints/
2. Descriptif : https://github.com/elyas-bhy/freya/wiki/freya-RESTful-API
3. Version déployée de l'API : http://goo.gl/o9GGBB

4 | Bibliothèque client

Un autre avantage de GCE est qu'il permet de générer automatiquement des bibliothèques clients multi-plateformes (Java/Android, iOS, JavaScript) afin de faciliter l'interaction avec l'API. La bibliothèque client consiste en deux parties différentes :

Le modèle client : représentation en Java des entités manipulables. Ces classes représentent le modèle existant du backend. Elles étendent la classe GenericJson afin d'être sérialisable sur le réseau. Exemple : le client dispose de la classe Artist (et son agrégat ArtistCollection [1]), générée par GCE. Cette classe est une représentation fidèle à celle du backend.

Builders et stubs : servent à faciliter la communication avec le backend. La classe générée Freya (nom de l'application) met en place une interface facile à utitliser pour le client afin d'exécuter des méthodes de l'API.

Code 4.1 – Exemple d'utilisation de la bibliothèque client

```
1 Artwork a = freya.artworks().get(artworkID).execute();
```

1. Des classes de type agrégats sont générés aussi car on peut sérialiser que des objets de type "bean", c.à.d avec des getters/setters. Les objets de type List<Object> ne sont pas conformes et doivent donc être encapsulés dans une classe ObjectCollection.

5 | Front-end

Pour mettre en place le front-end de Freya[1], nous avons voulu profiter de la bibliothèque client générée pour pouvoir accéder à l'API directement, sans passer par des étapes intermédiaires. C'est la raison pour laquelle nous avons choisi d'utiliser des pages JSP plutôt qu'utiliser GWT, qui est assez lourd à mettre en place et très verbeux.

5.1 Site web

Le front-end de notre projet est un site Internet constitué de pages JSP. Ce site est organisé de la manière suivante :

- A la racine du site (*webapp*), un fichier *index.jsp* sert de point d'entrée par défaut dans l'application.

- Un dossier *includes* regroupe les pages qui ne sont pas destinées à être affichées seules mais qui sont là pour factoriser du code, notamment les parties statiques du site (header, footer), mais aussi les manières de lister des éléments.

- Pour chaque entité, un dossier contenant les fichiers *view.jsp*, *edit.jsp* et *list.jsp* qui leurs sont propres. Le fichier *view* permet d'accéder à la vue détaillée d'une entité, permettant de consulter l'ensemble de ses champs. Le fichier *edit* permet de créer et modifier une entité en renseignant ses champs. Le fichier *list* permet de lister l'ensemble des entités d'un même type. Pour cela il utilise le fichier à inclure sachant lister une sous-partie ou l'ensemble de ces entités.

1. Front-end Freya : http://freya-app.appspot.com/

5.2 JavaScript

Nous avons eu recours à plusieurs bibliothèques JavaScript, notamment JQuery et ses modules DataTables, Chosen et Tabs. Nous avons choisi ces plugins afin de rendre l'interface plus agréable à utiliser.

Le plugin DataTables nous permet d'ajouter beaucoup de fonctionnalités à nos tableaux. Notamment la pagination des données, et la possibilité de trier le tableau suivant les colonnes. De plus, DataTables peut être appliqué sur un tableau HTML existant, ou encore nous pouvons lui passer en entrée des données en JSON, et lui demander de construire la table lui-même.

Chosen est un plugin qui s'applique aux menus déroulants, permettant d'effectuer des recherches au sein des menus, tout en appliquant une modification esthétique agréable.

Le plugin Tabs est un outil de tabulation de div. Avec ce plugin nous avons pu rendre plus agréable la disposition des pages présentant plusieurs tables. Avec ce plugin, nous pouvons également spécifier, à l'aide d'un suffixe à l'URL, la tabulation à afficher en priorité au moment du chargement de la page.

6 | Difficultés

La mise en place de ce projet n'a pas été sans contraintes :

— Maven : conflits entre versions des dépendances
— Restrictions GAE et comportements inattendus
— Contraintes NoSQL
— Prise en main de l'API Criteria
— Mise en place et maintien du cache (backend)
— Développement du front-end