Institut Polytechnique de Bordeaux – ENSEIRB-MATMECA  
Institut des Techniques d'Ingénieur de l'Industrie d'Aquitaine  
Formation d’ingénieur Réseaux et Systèmes d’Information

**BAYOU Mathieu**

**GARBAGE Maxime**

**Application répartie**

**« Réservation aérienne»**

**Remis le 16 mai 2016**

Sommaire

Erreur ! Le signet n’est pas défini.

Figures

Figure 1 - Schéma de séparation en couche 4

Figure 2 - Relation URL, méthode et action 4

Figure 3 - Pourcentage de nouvelles API ne supportant que le format JSON 5

Figure 4 - Pourcentage d'API avec un support XML 6

Introduction

L’évolution de l’informatique a conduit à faire évoluer le type d’application le plus rependu. Quelques années au part avant le standard résidé dans l’instauration d’application monoposte, dépende d’action d’un seul et unique utilisateur. Les technologies évoluant, la demande utilisateur s’est transformé pour aujourd’hui être orienté uniquement vers des application « collaborative » avec divers utilisateurs, des données accessibles depuis n’importe où et n’importe quand.

Afin de comprendre ce nouveau type d’application, nous avons réalisé un server de réservation aérienne. Ce document détail les choix techniques que nous avons effectué pour réaliser ce projet en utilisant comme vous le verrais des technologies et des concepts d’aujourd’hui.

1. Présentation du projet

Le projet consiste à proposer un service de réservation aérienne. Il devra répondre au problématique du client et ainsi lui permettre de :

1. **Recherche** - recherche de vols entre les aéroports de départ et d’arrivée, qui partent à la date donnée (pour plus de simplicité́, seul les vols aller simples sont considères). La réponse du service est une liste de dossiers contenant les informations suivantes pour chaque vol correspondant : numéro de vol, le nombre de places disponibles et le prix.
2. **Réservation** - réserver un billet pour le vol avec le numéro donné à la date donnée. Si la réservation est bien faite, le service doit renvoyer une ID de réservation. (Une ID de réservation est une série unique de chiffres et lettres génères lorsqu’un paiement est effectué pour une réservation).
3. **Annulation** annule la réservation avec l’ID donnée. En cas d’erreur, comme une réservation d’un vol non-existant ou une annulation de non réservation existante, le service doit informer les clients avec des messages appropries. L’annulation de la réservation entraine la restitution de la place au pool de places disponible.
4. Architecture technique

# Le choix de la technologie

Pour la réalisation nous avions le choix entre deux technologie, le Java et le C, offrant tous les deux des approches radicalement du développement un choix a dû être fait pour ainsi répondre au mieux aux attentes.

Aillant tout deux travaillés dans le domaine des applications web, nous avons utilisé notre expérience pour réfléchir au meilleur choix. Nous avons constaté que les problématiques d’un projet à un autre son régulièrement les même :

* Etre capable d’intercepter des requêtes
* Pouvoir lire et interpréter des formats de données (comme le Json)
* Pouvoir renvoyer des données sous un certain format
* Gérer les appels concurrents
* Gérer la sécurité
* Gérer les performances

Depuis l’arrivée du web et l’explosion des application répartie ces problématiques n’ont que légèrement évolué. C’est pourquoi de nombreuse librairie existe afin de répondre à ces points. Afin de nous décider entre ces deux technologies nous avons cherché à connaître lesquelles de ces langages disposés du plus de support dans ce domaine, mais surtout nous nous sommes renseigné sur les utilisations de ces derniers et quel été leurs part d’utilisation dans le monde en 2016. Il s’est avéré que depuis quelques années le C était en net recul face au Java qui se démarque largement : en effet en 2016 le JAVA à fait un bon 4,09% d’utilisation dans le monde alors que le C est en recul de 3,62% pour attendre respectivement, 20,95% d’utilisation pour le JAVA et 13,22% pour le C. Compte tenu de ces dernières informations il nous a semblé évident que dans l’intérêt de nos compétences future nous devions réaliser ce projet en JAVA.

# Pattern de développement

## Séparation en couche

Compte tenu de notre choix nous avions la possibilité d’architecturer notre projet en utilisant des patterns de développement permettant une meilleur évolutivité et lisibilité. Dans un premier temps nous avons mis en place une séparation en couche permettant de ne pas mélanger les entités de base de donnée, les traitements métiers et la couche de présentation (API).

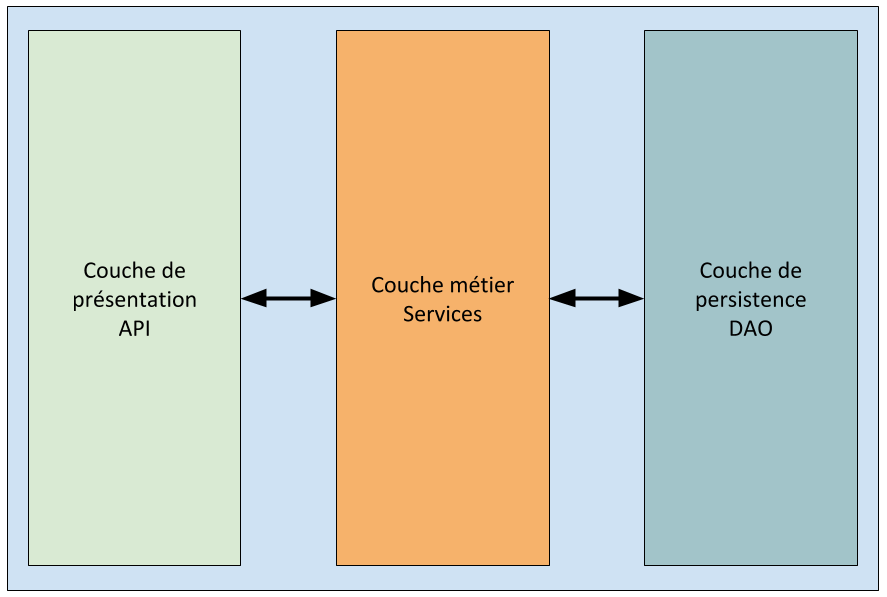


Figure - Schéma de séparation en couche

## Command/Query

Nous avons choisi l’utilisation d’un second pattern de développement qui consiste à diviser les traitement métier en 2 partie :

* **Command** : Service allant altérer les données.
* **Query**: Service allant récupérer des données.

Cela permet une meilleure structuration du projet et ainsi de ne pas se perdre dans les traitement métier.

# Choix des standards de l’API

## RESTful, de nombreux avantage

Nous avons opté pour l’utilisation du design d’API RESTful pour diverse raison :

* **Intuitivité**: REST se rapproche au plus près des standard http, il utilise les méthodes disponibles dans le standard pour effectuer toute les actions de base tel que l’ajout, la suppression, la mise à jour ou encore la récupération de données.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| URL | POST  créer | GET  lire | PUT  Mis à jour | DELETE  suppression |
| /articles | Crée un nouvel article | Récupère la liste d’article | Met à jour tous les articles | Supprime tous les articles |
| /articles/12 | Erreur | Récupère le détail d’un article | Met à jour l’article s’il existe  Sinon erreur | Supprime l’article s’il existe  Sinon erreur |

Figure - Relation URL, méthode et action

* **Performance** : REST est « stateless », c’est à dire qu’entre deux appels aucun état du client est gardé en mémoire sur le serveur. Cela permet de maintenir les performances en ne surchargeant pas l’utilisation des ressources.
* **Economie** : REST n’impose pas de format d’échange, contrairement à SOAP qui oblige l’utilisation d’un format contraignant, lourd, et avec une faible quantité de « charge utile » en son sein. En effet les enveloppes SOAP dispose de balise superflu mais nécessaire au standard afin d’être compréhensible. REST quant à lui permet l’utilisation de format tel que le JSON qui offre un très bon rapport entre la charge utile et le poids totale des données envoyé.

## JSON, format objet

Depuis notre premier choix technique, le JAVA, nous avons emprunté le chemin de la programmation objet, pour rester cohérent avec nos précédents choix nous avons opté pour le format JSON qui permet une représentation objet des données aisément interprétable par l’homme. Le XML aurait pu être un concurrent mais compte tenu de son architecture en balise, beaucoup d’élément son donc répété alors qu’il n’apporte pas d’information supplémentaire lors d’un échange. De plus l’utilisation mondiale du format XML est en perpétuel baisse et ne laisse pas présager une grande expansion de l’intégration de ce dernier. Toujours dans un souci d’utiliser des technologies en accord avec l’évolution du monde informatique et son état actuel il ne nous a pas semblé approprié d’utiliser ce format.

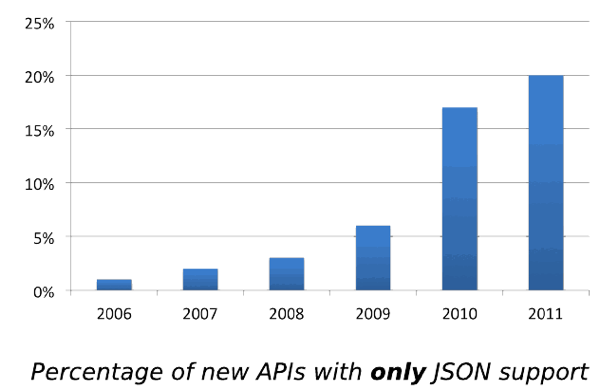


Figure - Pourcentage de nouvelles API ne supportant que le format JSON

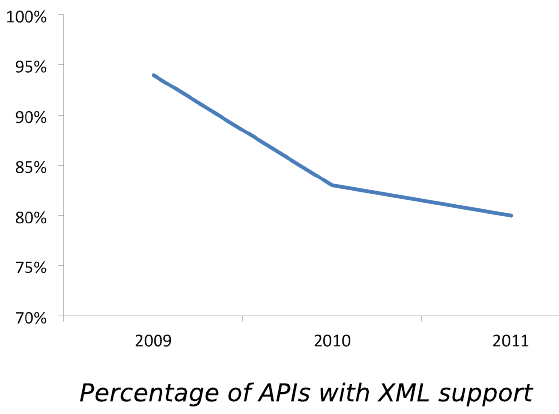


Figure - Pourcentage d'API avec un support XML

1. Fonctionnalité