09/01/2014

Projet XML

Navarre Julien, Pavinich Christian, Coda Stephen

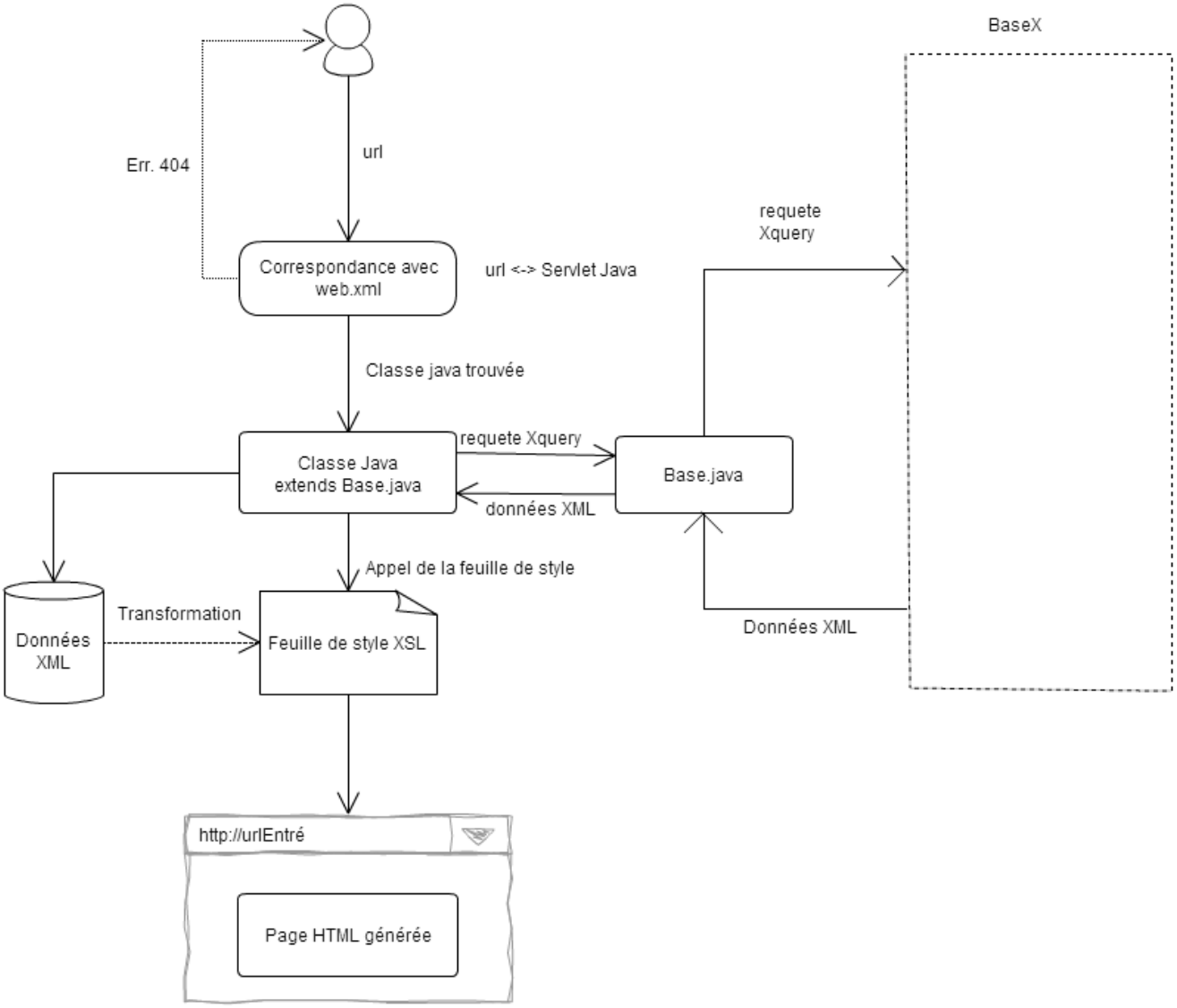
Sommaire

I. Environnement de développement

Ce projet se résume donc à une application web exploitant les données fournies sous format XML. Pour pouvoir les utiliser nous avons installé le moteur de bases de données XML baseX couplé à son API java pour l'utilisation dans un serveur Tomcat.

Nous allons donc rappeler le fonctionnement d'une application web développée sous Tomcat, avec en addition l'architecture adoptée pour notre projet.

A. Schéma



B. Explications

Le fonctionnement de notre application suit donc le cycle de vie d'une application Tomcat classique. Pour fonctionner avec des données XML nous avons donc fait hériter chacun de nos servlets par la classe Base.java, qui représente simplement le modèle de notre application. Il permet de se connecter à BaseX, et d'effectuer des requêtes Xquery afin de récupérer les données XML nécessaires.

Au vu du cycle de vie d'un servlet nous avons choisi de faire hériter ceux-ci par Base.java. Il n'était pas nécessaire de faire de Base.java un singleton ou une classe statique.

Une fois les données XML récupérées, le servlet fait simplement appel à la feuille de style XSL pour faire la transformation (grâce à l'objet Transformer) avec les données pour donner la page qui sera affichée sur le navigateur.

C. Outils utilisés

Pour ce projet nous avons utilisé les logiciels suivants :

- Netbeans avec serveur Apache Tomcat
- BaseX
- SourceTree (gestionnaire de repository Git)

Et les langages utilisés dans le développement sont :

- Java
- XQuery, XSLT, XSLFO
- HTML, JavaScript, Jquery, Ajax, CSS

Enfin, nous avons utilisé plusieurs plugins et librairies pour nous aider à développer les fonctionnalités recherchées :

- Bootstrap
- DataTables
- GoogleMaps JavaScript API
- d3, nvd3 (plugins SVG)

II. Fonctionnalités

Chaque fonctionnalité du projet va donc correspondre à un Servlet et une feuille xsl et aura un rôle unique qui peut varier selon les paramètres entrés dans l'url.

Nous noterons qu'il y a quelques éléments statiques dans notre projet, qui sont les classiques header et footer qui permettent la gestion des imports de scripts et de feuilles de styles css, et évidemment la barre de menu. Ceci est géré grâce à la directive xsl call-template qui doit être précisée sur chaque fichier xsl.

A. Liste

La première fonctionnalité développée est la possibilité de lister simplement les hôtels dans une table html avec les critères principaux d'un hôtel. C'est-à-dire le nom, l'adresse, le numéro de téléphone et l'adresse e-mail de chaque hôtel.

Ce tableau, grâce au plugin dataTables fait en JQuery nous permet de le gérer dynamiquement. Ainsi, nous pouvons gérer le nombre d'hôtels affichés, les trier par colonne (ordre croissant ou décroissant), et même ajouter des critères grâce à des input. Dans notre cas nous n'avons ajouté qu'un filtre sur le prix avec un input de type text (surcharge de la méthode javascript push de DataTables pour l'évènement) ; car il était facile d'en ajouter d'autres, et pas très pertinent pour le but du projet. On peut donc imaginer de nombreuses possibilités d'affichage avec ce tableau en ajoutant d'autres filtres.

De plus, nous avons intégré l'option d'export de DataTables qui permet donc l'export de la table sous différents formats intéressants : pdf, Excel, csv, copier directement son contenu dans le presse papier ou encore l'imprimer.

Enfin, un lien a été ajouté sur le nom de chaque hôtel afin d'accéder à son détail via une autre page/fonctionnalité de l'application (Servlet Detail).

B. Détail

La deuxième fonctionnalité développée est le détail d'un hôtel. En passant l'id d'un hôtel en get dans l'url, le Servlet Detail récupère ensuite toutes les informations de cet hôtel pour pouvoir les afficher dans une page.

Cette page va présenter les informations les plus intéressantes pour un hôtel en commençant par le nombre d'étoiles, et en disposant les images dans un Carrousel (plugin OwlCarrousel). Ensuite, chaque information est placée dans un menu avec des div déroulants de type Collapse pour ne pas inonder la page quand l'utilisateur visualise plusieurs informations les unes à la suite des autres.

C. Carte

Ensuite, la prochaine fonctionnalité abordée est l'affichage des hôtels sur une carte GoogleMaps. Ceci a été fait avec l'API JavaScript pour GoogleMaps.

Les hôtels sont donc tous affichés sur cette carte, avec la possibilité de cliquer sur un marqueur pour rediriger l'utilisateur vers la page de détail de l'hôtel cliqué (avec la méthode JavaScript de l'API addListener). Notons qu'au retour sur la carte le marquer cliqué devient actif (rebondis) pour ne pas que l'utilisateur soit confus par rapport à quel hôtel avait été sélectionné.

Nous avons pensé que dans la recherche d'hôtel l'information primordiale et déterminante, est le prix. C'est pourquoi nous avons mis un filtre sur la liste avec un prix maximal.

Une autre dimension très importante est la localisation, nous avons donc jugé pertinent d'ajouter une information sur le prix en fonction de la zone géographique. A côté de la carte, un histogramme renvoi les prix minimums pour une chambre simple parmi les hôtels affichés sur la carte. Seuls les prix des hôtels dans les frontières de la carte sont représentés. La carte s'actualise à chaque fois qu'on navigue (zoom ou drag). Cela permet une visualisation rapide des prix dans la zone regardée. On peut se rendre compte facilement si les prix d'un hôtel sont élevés ou pas par rapport aux autres hôtels de la zone grâce au prix moyen et à la couleur nuancée des barres. Pour mieux savoir de quel hôtel il s'agit, au survol des barres on affiche le nom, et animons le marquer représentant l'hôtel sur la carte. Les barres sont cliquables et renvoient sur la page "Détail" de l'hôtel. Le graphique au format SVG a été réalisé grâce la librairie d3.js. La fonction est construite à partir d'un listener sur l'événement "bounds_changed" de l'API GoogleMaps. Lorsque l'événement "bounds_changed" est détecté, nous récupérons les coordonnées (x,y) des 4 angles de la carte puis nous trions dans les données XML les hôtels dont la latitude et la longitude est comprise dans les limites de la carte, et c'est à partir de ce nouveau jeu de données que le graphe est construit.

D. Graphiques

En plus du graphique de la carte, nous avons réalisé deux autres graphiques SVG dans l'onglet "Graphes" (Servlet Graphes). Un premier graphique circulaire représente la répartition des hôtels en fonction de leur nombre d'étoiles. Et un second graphique représente le prix minimum moyen d'une chambre en fonction du nombre d'étoiles.

Ces deux graphiques ont été réalisés avec la librairie nvd3 qui permet de simplifier la création de graphiques basés sur d3.

L'utilisation de librairies simplifie réellement la production de graphiques SVG. Par exemple, cela évite toutes les manipulations nécessaires à la remise du repère orthonormé à l'endroit.

Nous avons tout de même réalisé un dernier graphique SVG sans utiliser de librairies, afin de permettre sa transformation en PDF. Il est donc visible dans notre exemple d'export PDF (Servlet Pdf). Afin de produire ce graphique à l'intérieur de notre code XSLFO, il a tout simplement fallu inclure le code SVG à l'intérieur d'une balise permettant l'intégration d'éléments SVG "<fo:instream-foreign-object xmlns:svg="http://www.w3.org/2000/svg">" et préciser <svg:balise> à l'intérieur de chaque balise relative à SVG.

III. Problèmes rencontrés

Lors du développement de ce projet nous avons eu plusieurs problèmes qui sont intéressants à noter :

- Difficulté de la mise en place de l'environnement de développement

Ce projet a été pour nous l'un des plus difficiles à commencer au vu du nombre plutôt important de technologies à mettre en place et à faire fonctionner ensemble, et ce, avant même de pouvoir commencer à travailler.

Le choix entre BaseX et ExistDB n'a pas été évident au début, l'utilisation et la configuration encore moins. Mais notre choix s'est porté vers BaseX qui nous permettait une meilleure flexibilité par rapport à Java et l'API qui l'accompagnait.

Ensuite, le fonctionnement de Tomcat avec les feuilles de style xsl présentait des erreurs déroutantes par rapport aux caractères qui étaient acceptés par Tomcat, et requis pour coder les directives xsl. Une compréhension de la console Tomcat est devenue rapidement importante. Ce problème était encore plus gênant lors de l'utilisation de code JavaScript.

Il est donc facile de dire que l'étape la plus difficile du projet a sans doute été la mise en place de l'environnement.

- Organisation/Séparation efficace du code

Le besoin d'utiliser des boucles/appels xsl dans le code javascript a rendu impossible l'export de celui-ci dans des fichiers séparés, ou du moins très peu envisageable (notamment pour la carte googleMaps). De ce fait, on imagine que pour un projet de plus grande envergure les fichiers xsl risquent d'être pollués par du script Javascript.

Ensuite, la gestion de templates n'est pas très pratique en xsl, car pour chaque fichier xsl nous sommes obligés d'appeler le template xsl, de plus, nous ne pouvons surcharger celui-ci pour l'adapter à une fonctionnalité, ou définir/importer des styles css pour une page unique. Une comparaison facile serait le framework Symphony qui gère plus efficacement ce problème-là.

Il existe certainement des solutions optimales à ces problèmes qui viennent surtout de notre manque d'expérience avec cette combinaison de technologies. Avec le temps que nous avons pu investir dans ce projet, nous n'avons pas eu l'occasion de trouver de meilleures solutions à ces problématiques.