Spécifications techniques

Trajan Bronner
David Courtinot
Gabriel Guiral

11 août 2014

Sommaire

Chapitre I

Dépendances externes

L'application s'appuie sur diverses librairies open-source, au cours du développement il n'était pas important de s'intéresser aux termes exacts de leur licence mais cela le deviendrait lors d'une éventuelle mise en production. Nous référençons donc toutes les librairies utilisées pour facilitier ce travail ainsi que donner une vue d'ensemble sur les dépendances externes.

I .1 Côté client

Cela concerne principalement la page "Personnaliser mon biscuit" qui fait appel à du code Javascript externe.

- Three.js (WebGL, rendu 3D)
- Kinetic.js (Visualisation 2D du biscuit et positionnement des objets)
- JQuery 1.10 + Bootstrap 3.2 (Utilisé dans tout le site pour le design)

I .2 Côté serveur

- Joda time (utilisation très légère pour gérer le formattage des dates dans l'application)
- une API de génération de QR code (accessible par l'URL http ://www.esponce.com/api/v3/generate)

I .3 Application bureau

- JDOM (pour la génération des templates SVG)
- Controls FX 8 (boîtes de dialogue)
- Joda time (même chose que dans l'application de web)
- mysql-connector-java-5.1.28-bin.jar (ce n'est pas une librairie, mais il est nécessaire de l'ajouter au CLASSPATH pour permettre la communication avec la base de données)
- une API de génération de QR code (accessible par l'URL http ://www.esponce.com/api/v3/generate)
- une API de génération de QR code (accessible par l'URL http ://api.qrserver.com/v1/create-qr-code/)

Chapitre II

Pré-requis et configuration

Ce chapitre concerne uniquement le côté serveur de l'application web et l'application bureau, la côté client du site fonctionnera quant à lui a priori sur n'importe quel navigateur *suffisamment récent.*

II .1 Côté serveur

Nous avons utilisé le serveur GlassFish 3, fournissant de façon native une implémentation (Mojarra 2.0) du framework MVC Java Server faces (JSF). Pour la persistence des données, nous avons utilisé JPA, EclipseLink, et MySQL. Utiliser Hibernate plutôt qu'EclipseLink entraînerait la modification de quelques lignes dans le fichier persistence.xml. Un changement de base de données pourrait requérir la réécriture de certaines requêtes SQL dans l'EJB UserManager. Il est également nécessaire de définir un pool de connexions. Nous fournissons le fichier *domain.xml* complet avec ce document pour plus d'informations. Enfin, l'environnement d'exécution est JRE 7.

II .2 Application bureau

Pour faire tourner cette applicaiton écrite en JavaFX 8, il est nécessaire de disposer une machine virtuelle Java (JRE 8 ou ultérieur).

Chapitre III

Explications techniques sur certains aspects

III .1 Application web : interactions client-serveur

Cette section s'attache uniquement à décrire les interactions client-serveur pour la page de personnalisation des biscuits, dans la mesure où une grande partie du code est en Javascript. Tout a été fait de sorte à fournir une "interface" entre client et serveur pour cette page de sorte qu'il ne soit pas nécessaire de réécrire le client en cas de changement de langage serveur.

Le serveur est chargé de :

- remplir les champs cachés contenant les informations sur le biscuit
- se tenir prêt à rafraîchir ces informations en répondant à un appel Ajax du client lorsque la valeur du champ de sélection du biscuit est modifiée côté client
- traiter le formulaire créé dynamiquement par le client lorsque celui-ci lui est transmis
- fournir une API permettant de générer un QR code à partir d'une taille (actuellement comprise entre 1 et 20) et de son contenu (actuellement dépourvu d'accents et d'espaces). Cette API doit être accessible par simple URL et peut éventuellement n'être qu'un proxy côté serveur appelant lui-même une API externe (c'est le cas ici comme nous en reparlerons).

De son côté, le client écoute les changements de valeur de ce même champ et déclenche une attente de 500 ms afin d'attendre la réponse du serveur, puis rafraîchit le rendu 3D et la visualisation 3D du biscuit. S'il apparaît que le serveur ne garantit pas un temps de réponse inférieur à 500 ms, il faudra envisager d'augmenter ce délai d'attente côté client voire d'opter pour une approche plus fiable (maais plus complexe). Enfin, le client part du principe que les fichiers de texture et autres ont le même nom que la référence de l'objet en base de données (par exemple si le biscuit est référencé par *princeChoco*, la prise de vue 2D devra s'intituler *princeChoco2D.png*, le fichier de texture devra s'appeler *princeChoco.png* et enfin le maillage de l'objet devra être nommé *princeChoco.obj*.

III .2 Application bureau : génération des templates SVG

Il faut fournir un certain nombre de fichiers de description des biscuits et de leur disposition sur la plaque pour permettre la génération des templates svg. Supposons que l'on souhaite introduire un biscuit dont la référence (celle qui l'identifie dans la base de données) est princeChoco. Il va falloir pour cela définir

plusieurs fichiers :

- princeChoco. properties
- princeChoco_template.svg

Le premier de ces deux fichiers est un simple fichier de propriétés qui doit contenir les propriétés suivantes (toutes les longueurs sont à exprimer en mm) :

- **repeat.x** : nombre de biscuits sur la plaque dans chaque ligne
- **repeat.y** : nombre de biscuits sur la plaque dans chaque colonne
- **repeat.spacing** : espacement vertical et horizontal entre chaque répétition du motif
- **pattern.edge** : longueur du côté du carré circonscrit au motif. Si besoin, cela pourra être divisé en pattern.edge.x et pattern.edge.y pour utiliser des motifs contenus plutôt dans des rectangles.
- **pattern.center.x** : abscisse du centre du carré circonscrit au premier motif en partant du haut, gauche
- **pattern.center.y** : ordonnée du centre du carré circonscrit au premier motif en partant du haut, gauche
- **custom.edge** : même chose que pattern.edge mais cela concerne la zone éditable et non le biscuit entier
- **custom.center.x** : même chose que pattern.edge mais cela concerne la zone éditable et non le biscuit entier
- **custom.center.y** : même chose que pattern.edge mais cela concerne la zone éditable et non le biscuit entier

Enfin, le deuxième fichier est un fichier SVG décrivant la disposition des biscuits sur la plaque à découper. Il doit bien entendu respecter les valeurs indiquées dans le fichier de propriétés.

III .3 Améliorations à apporter

Définition des biscuits

Dans l'ensemble, nous avons résolu toutes les questions techniques qui se sont posées à nous. Cela étant, il existe quelques points où une solution plus optimale pourrait être utilisée à moindre effort. Citons par exemple la définition de la zone éditable sur un biscuit, se faisant à l'aide l'attribut *edgeLength* de la table BISCUIT. Il s'agit de la longueur du carré délimitant la zone éditable sur le biscuit, centrée en le biscuit. Pour bien faire, il faudrait en réalité ajouter un attribut $center\_X$ et $center\_Y$ pour positionner le centre de la zone éditable ailleurs qu'au centre du biscuit (la forme du biscuit l'imposera parfois) et diviser *edgeLength* en $edge\_X$ et $edge\_Y$ pour permettre des zones d'édition rectangulaires, toujours pour s'adapter à la forme du biscuit.

Génération des QR codes

Nous utilisons actuellement deux APIs externes de génération de QR code et cela d'une part complique le code, d'autre part nous met à la merci d'un éventuel arrêt du support d'une de ces API ou de son dysfonctionnement. La raison pour laquelle nous utilisons ces APIs est expliquée dans le commentaire de code suivant :

> *This piece of code deserves an explanation. It's quite twisted because we are dependent of two QR code APIs which have both their pros and cons. The first API let us use a transparent background but generates an ugly SVG. At the contrary, the second API forces us to remove a*

white background but generates a nice SVG. Then, we have decided to use the first API client-side to limit the complexity of the JS code, and finally use the second API to actually generate the template. As these APIs don't use the same standard to designate the size of the QR code, we need to convert it.

Pour faire au mieux, il faudrait implémenter notre propre API avec le langage serveur choisir, ce qui ne devdrait pas être difficile si Java est conservé côté serveur (existant de nombreuses librairies facilitant la tâche).

Options de personnalisation

Nous aurions pu aller plus loin dans la personnalisation des biscuits, en permettant par exemple de choisir la couleur ou la police (pour les textes), voire de rajouter des images. Nous n'avons pas implémenté ces fonctions car nous savions que la technologie de gravure laser ne le permettait de toute façon pas (hormis pour les polices), mais si cela devient physiquement possible il devait être facile de l'intégrer dans notre application, qui a été conçue de telle sorte que lui apporter des extensions raisonnables soit aisé.