### Plate-forme web de gestion d’un carnet d’adresse

Equipe :

- BAULIN Vincent

- BARON Julien

- KOUMIRA Adnane

**L’architecture et les outils** :

La plateforme web a été réalisée en Java, à l’aide du Framework Spring MVC. Elle se base sur le pattern Model – Vue – Contrôleur, ainsi qu’un système de servlet représenté par le fichier mvc-dispatcher-servlet.xml. Nous avons eu pour objectif de réaliser une architecture composée de trois couches qui suit le principe d’inversion des dépendances. On retrouve donc deux dossiers (DAO et Service) qui se chargent de recueillir proprement des données depuis une base (une Map dans notre cas).

Le design de la plateforme s’appuie sur Bootstrap et sur du CSS classique. Un aspect dynamique a été apport à l’aide de jQuery, notamment pour le tableau récapitulatif des contacts.

**Les fonctionnalités :**

Il est possible de visualiser l’ensemble de ses contacte sur la page principale. Un champ recherche permet de retrouver un contact en fonction de l’ensemble de ses attributs (nom, prénom, email,…). Dans la barre de menu, nous pouvons accéder aux deux pages les plus importantes de l’application. Nous pouvons ajouter un utilisateur en renseignant les champs proposés. Un bouton permet de rajouter des adresses de livraison de manière dynamique. Malheureusement, nous n’avons pas réussi à réaliser la validation des renseignements relatifs aux adresses.

Depuis la page principale, nous pouvons aussi visualiser les caractéristiques de nos contacte, supprimer des contacts, ainsi que réaliser des modifications. Lorsque l’on accède à la page d’édition, nous pouvons voir l’ensemble des renseignements de notre contacte ainsi que ses adresses. Pour les même raison que précédemment, la validation du formulaire ne réalisera que l’enregistrement des informations hors adresses.

**Les tentatives ratées :**

Nous aurions pu séparer la création des contacts et des adresses, mais l’association de ces deux éléments n’aurai pas été ergonomique. Cela aurait rendu le processus lourd lors des modifications. Ainsi, nous avons essayé de générer dynamiquement les champs d’adresse limitée à deux adresses de livraisons. Pour réaliser la validation des adresses, nous nous somme intéressé à l’Ajax ainsi qu’au Json car Spring ne semblait pas pouvoir nous permettre de valider des Pojo contenant des listes puisqu’il se base sur les getters et les setters. Malheureusement, nous n’avons pas réussi à mener cette idée à terme.

Nous avons aussi dans une moindre mesure essayé d’introduite une google map dans la page de création et d’édition de contacte dans le but d’avoir une information visuel et dynamique. Nous n’avons pas eu le temps d’implémenter cette fonctionnalité malgré un très bon exemple disponible sur internet :

<http://universimmedia.pagesperso-orange.fr/geo/loc.htm>

**Transformation du projet maven en projet google app engine**

Dans le menu propriété du projet maven, aller dans l’ongle google -> App engine puis cocher« Use Google App Engine » et cliquer sur OK.

Ensuite, ajouter le fichier appengine-web.xml dans le dossier src/main/webapp/WEB-INF sous eclipse, et le compléter (voir exemple sur internet ou copier le fichier d’un projet google app engine).

Enfin déployer le projet sous Google App Engine.

Nous avons obtenu l’erreur suivante :

com.google.appengine.tools.admin.JspCompilationException: Failed to compile jsp files.

Exception in thread "main" org.apache.jasper.JasperException: The absolute uri: http://www.springframework.org/tags/form cannot be resolved in either web.xml or the jar files deployed with this application

at org.apache.jasper.compiler.DefaultErrorHandler.jspError(DefaultErrorHandler.java:51)

at org.apache.jasper.compiler.ErrorDispatcher.dispatch(ErrorDispatcher.java:409)

at org.apache.jasper.compiler.ErrorDispatcher.jspError(ErrorDispatcher.java:116)

at org.apache.jasper.compiler.TagLibraryInfoImpl.generateTLDLocation(TagLibraryInfoImpl.java:315)

at org.apache.jasper.compiler.TagLibraryInfoImpl.<init>(TagLibraryInfoImpl.java:148)

at org.apache.jasper.compiler.Parser.parseTaglibDirective(Parser.java:386)

at org.apache.jasper.compiler.Parser.parseDirective(Parser.java:449)

at org.apache.jasper.compiler.Parser.parseElements(Parser.java:1396)

at org.apache.jasper.compiler.Parser.parse(Parser.java:130)

at org.apache.jasper.compiler.ParserController.doParse(ParserController.java:255)

at org.apache.jasper.compiler.ParserController.parse(ParserController.java:103)

at org.apache.jasper.compiler.Compiler.generateJava(Compiler.java:185)

at org.apache.jasper.compiler.Compiler.compile(Compiler.java:347)

at org.apache.jasper.JspC.processFile(JspC.java:1192)

at org.apache.jasper.JspC.execute(JspC.java:1341)

at com.google.appengine.tools.development.LocalJspC.main(LocalJspC.java:35)

**Le diagramme UML :**

Voici un diagramme UML du projet :

