Maintenance Applicative d'une application développée en JAVA

Stage de 2ème année

BOISSEAU-SABLE, Arthur

BTS Services Informatiques aux Organisations
Spécialité SLAM

2015-2016

MAIRIE DE PARIS

Sommaire

Remerciements

Je remercie la ville de Paris de m'avoir accueilli lors de mon stage de seconde année de BTS, qui m'aura permis d'acquérir de l'expérience dans le milieu du développement mais également de découvrir, plus en profondeur, le travail en milieu professionnel.

Mon tuteur et responsable du pôle intégration du centre de compétences Facil'Famille, M.MENIVAL Christophe, fut très patient au cours de ce stage et a pu me prodiguer de précieux conseils tout au long de de ces deux mois. Son suivi m'aura permis de découvrir de nouvelles méthodes de travail, et je l'en remercie grandement.

Je souhaite également remercier M.OUEDRAOGO Mahamadi qui put m'apporter un soutien et une aide non négligeable lors de la réalisation des projets de développement qui m'ont été confié.

Enfin, je remercie Mme BALLEREAU Eva qui fut à mon écoute et pu m'aider à plusieurs reprises, ainsi que toute l'équipe du pôle intégration qui fut très agréable lors de ces 8 semaines.

1. Introduction

Ce document présente et détaille la réalisation des projets qui m'ont été confiés lors du stage.

Il est composé tout d'abord d'une rapide présentation de l'administration (la Mairie de Paris) et du centre de compétences Facil'Famille où j'ai réalisé mon stage.

Puis, nous présenterons rapidement l'application Centre De Loisirs (CDL) ainsi que la pop-up Base-Adresses dont la maintenance m'a été confié.

Nous aborderons ensuite de manière générale puis plus spécifique comment les projets informatiques ont été réalisés.

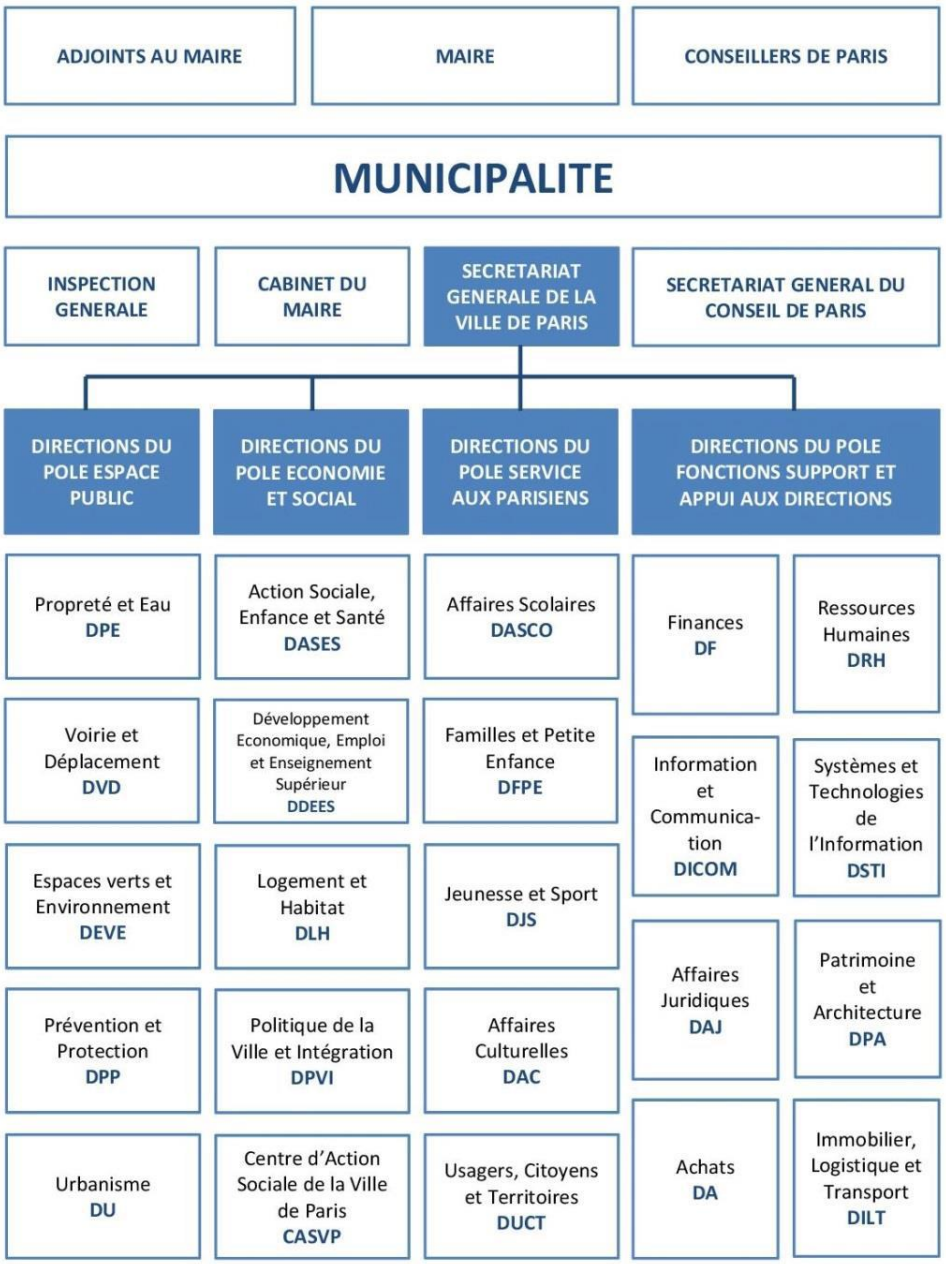
Enfin, nous nous pencherons sur les tests réalisés une fois l'application modifiée.

1.1. Présentation de l'administration

La Mairie de Paris est une des plus grandes administrations publiques de France. En effet, la ville est divisée en 20 arrondissements, chacun dirigé (administrativement) par un maire.

De plus, chaque arrondissement se voit élire des conseillers de Paris (au minimum 3 par arrondissement), siégeant au conseil de Paris. Cette assemblée, possédant les attributions d'un conseil municipal et d'un conseil départemental, est présidée par le maire de Paris.

La mairie de paris possède donc une structure assez large et complexe. En quelques chiffres, elle regroupe plus de 50 000 agents, 4 grands pôles, 23 directions et environ 3 000 sites :



1.2. Présentation du Centre de Compétences Facil'Famille

Le projet Facil'famille est un programme qui fut lancé en 2009. Ce dernier a pour but de moderniser les outils liés aux activités périscolaires, culturelles et touchant à la petite enfance. Il doit également faciliter la relation entre les usagers et la mairie de Paris pour la facturation et l'inscription.

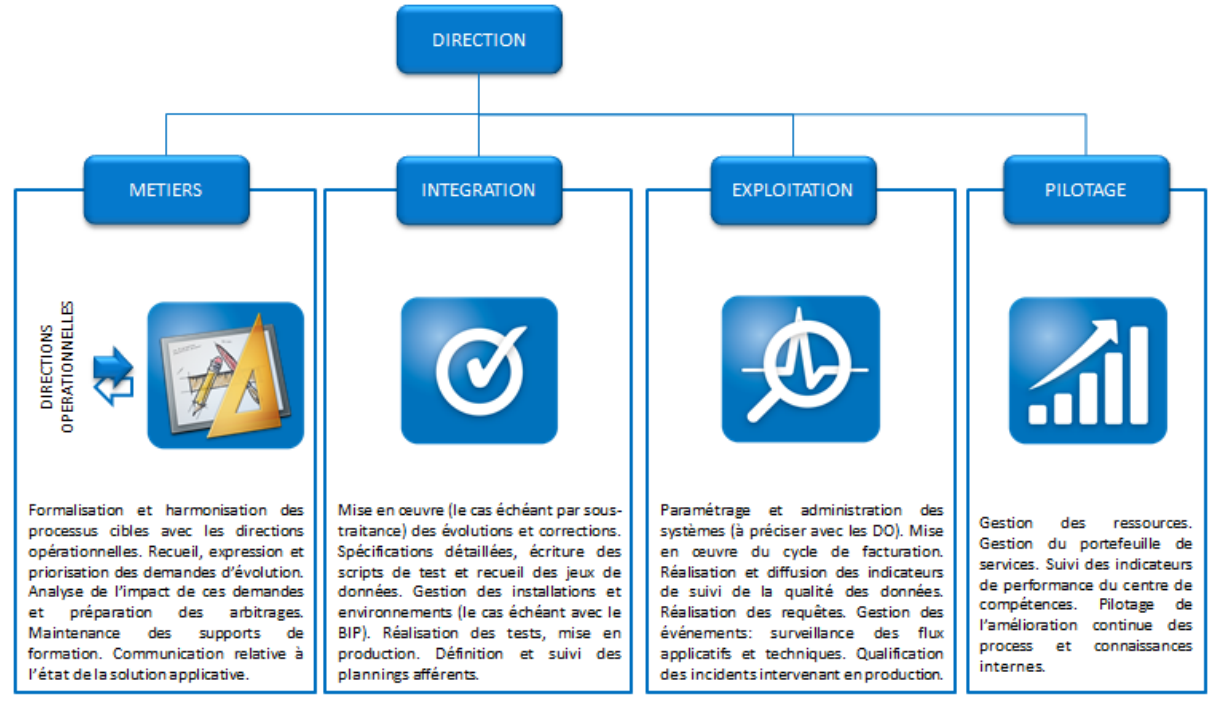
Ce centre se divise en 4 pôles :

- Le pôle intégration, qui gère la mise en production (correction et évolution), le déploiement et le test de services ainsi que l'écriture de spécifications détaillées et de script de test.

- Le pôle métier, qui gère les demandes d'évolutions et qualifie les incidents signalés par les directions qu'il redirige ensuite. Il a également pour rôle de prévoir les conséquences organisationnelles d'une évolution de système, ou encore de vérifier que les besoins exprimés soit en adéquation avec les objectifs stratégiques de la mairie de Paris.

- Le pôle exploitation, qui gère le paramétrage logiciel et la sécurité du service. Il met également en place des outils de suivi pour les directions opérationnelles, leur permettant de gérer la qualité des données renseignées par les utilisateurs.

- Le pôle pilotage, qui gère les ressources humaines et financières de Facil'Famille ainsi que l'offre des services et leur qualité. Il décide de la stratégie du centre de compétences et pilote l'amélioration continue des process.



2. Situation Professionnelle – Modification d'une application développée en JAVA connecté à des services web

2.1. Description et analyse de la situation professionnelle

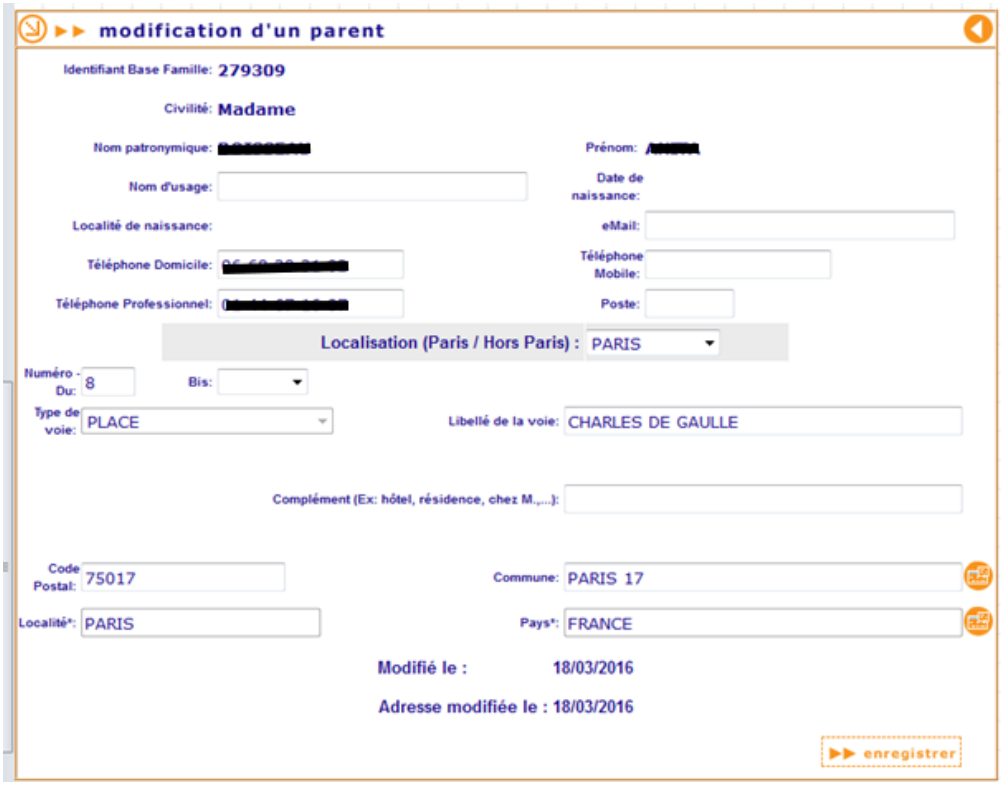
2.1.1. Contexte

CDL est une application permettant de saisir la présence des enfants ainsi que des animateurs chargés de les encadrer dans les centres de loisirs parisiens (les mercredis et pendant les vacances scolaires). Elle fut créée en 2003 et recense aujourd'hui 650 activités de centres de loisirs.

Elle donne également accès aux informations concernant les enfants (et leurs parents) fréquentant les écoles publiques de la ville de Paris ou ayant déjà fréquentés un des centres.

De plus, elle dispose d'une fonctionnalité permettant de modifier ces informations. Il est donc possible, par exemple, d'y renseigner une adresse mail ou de la modifier.

L'utilisateur a également la possibilité de renseigner ou modifier une adresse.



Dans cette situation, cdl fait appel à une pop-up proposant de rechercher une adresse dans la ville de Paris.

Or cette pop-up ne dispose que d'un seul de critère de recherche.

2.1.2. L'existant

L'application existante est une pop-up de sélection d'adresses. Cette dernière est ouverte sur le navigateur par une fonction Javascript qui prend en paramètre une adresse saisie par l'utilisateur. La pop-up sollicite ensuite une application J2EE autonome de sélection.
Cette application :

Interroge, en mode configurable http ou https, le service d'adresses de Base Adresses en transmettant l'adresse saisie par l'utilisateur.

Contrôle et trie la liste d'adresses retournées par Base Adresses, puis construit un formulaire de sélection d'adresses.

Une fois l'adresse sélectionnée, transmet à la fenêtre principale, par une autre fonction Javascript, les informations relatives à l'adresse sélectionnée par l'utilisateur.

2.1.3. Le besoin

La pop-up ne possède qu'un seul critère de recherche, l'adresse. Cela peut engendrer des problèmes lors de l'affichage des résultats, qui peuvent être trop nombreux. De plus, l'utilisateur peut avoir besoin d'affiner sa recherche.

L'ajout d'un critère arrondissement à la pop-up aura donc pour principal objectif la récupération et le traitement d'une nouvelle valeur saisie par l'utilisateur, en plus de l'adresse.
Cela permettra :
- De retourner une liste d'adresses plus pertinente, plus précise en résultat d'une recherche.
- De limiter le déclenchement d'erreurs liées à un trop grand nombre de résultats.

2.1.4. Démarche de résolution

Les différentes étapes de ce projet furent réalisées dans l'ordre suivant :

- Étude de la spécification fonctionnelle de l'application.

- Installation du serveur Tomcat et des outils de travail (eclipse,..).

- Installation de l'application (import du fichier .war sur eclipse, ajout des librairies/connexion au service web, connexion de l'application au serveur).

- Analyse du code source de l'application.

- Analyse des librairies du service web.

- Rédaction compte-rendu sur les librairies du service web, après documentation.

- Localisation des classes à modifier.

- Ajout du nouveau champ de saisie 'Arrondissement' à la popup.

- Ajout de lignes de code permettant la récupération des informations saisies dans ce nouveau champ.

- Ajout de lignes de code permettant de traiter l'information (création d'une nouvelle méthode, modification de fonctions existantes, intégration de ce nouveau critère lors de l'appel du service web,…).

- Rédaction d'un compte-rendu sur la modification de l'application.

2.2. Productions

2.2.1. Documentation : les librairies du service web

Avant de passer aux projets informatiques, il m'a été demandé de me documenter sur les librairies nécessaires à l'utilisation du service web Base Adresses. Cela m'aura permis de mieux comprendre son fonctionnement, ce qui me fut très utile lors du développement.

Activation.jar :

Ce fichier d'archive Java contient JavaBeans Activation Framework Specification, un Framework permettant de déterminer le type d'un morceau de données et d'en encapsuler l'accès. Il détermine ensuite les opérations réalisables et les met à exécution. Par exemple, si un navigateur affiche une image en JPEG, le Framework lui permettra de l'identifier en tant que tel puis de la localiser et enfin d'instancier un objet qui permettra de la modifier.

Source : http://www.oracle.com/technetwork/articles/java/index-135046.html

Axis-ant.jar/Axis.jar :

Ces fichiers permettent de fournir un environnement d'hébergement de service web, ainsi qu'un ensemble d'outils de développement facilitant la création, le déploiement et le test de ces services. Il permet également l'accès à des services tiers.

Sources : *http://www-iqm.univ-mlv.fr/~dr/XPOSE2003/axis_seng/introduction.html*

https://fr.wikipedia.org/wiki/Apache_Axis

Axis-jaxrpc.jar/jaxrpc.jar :

Ce fichier d'archive est une API permettant de transmettre des messages SOAP en mode RPC. Plus clairement, il facilite la communication et les échanges (appel de méthodes/fonctions,…) entre services web et applications.

Sources : *https://fr.wikipedia.org/wiki/JAX-RPC*

http://searchsoa.techtarget.com/definition/JAX-RPC

Commons-beanutils.jar :

Ce fichier permet de faciliter l'utilisation de l'API Réflexion (qui permet d'accéder aux caractéristiques d'une classe : introspection, mais aussi d'instancier des classes de manière dynamique,…).

Sources : *http://ricky81.developpez.com/tutoriel/java/api/reflection/#L2*

http://www.jmdoudoux.fr/java/dej/chap-apache_commons.htm

http://commons.apache.org/proper/commons-beanutils/

Détail : http://commons.apache.org/proper/commons-beanutils/javadocs/v1.8.3/apidocs/

Commons-codec.jar :

Ce fichier propose des algorithmes d'encodage et de décodage.

Sources : *http://www.jmdoudoux.fr/java/dej/chap-apache_commons.htm*

http://commons.apache.org/proper/commons-codec/

Commons-collections.jar :

Ce fichier contient différentes collections fournissant de nouvelles interfaces, implémentations et utilités lors de la construction des classes (Bag interface : pour les collections qui ont un certain nombre de copies de chaque objet, MapIterator interface : fournit une itération simple et rapide pour les cartes, …).

Sources : http://commons.apache.org/proper/commons-collections/

http://www.jmdoudoux.fr/java/dej/chap-apache_commons.htm

Commons-discovery.jar :

Ce fichier propose un service de localisation de ressources (classes).

Sources : http://www.jmdoudoux.fr/java/dej/chap-apache_commons.htm

http://commons.apache.org/proper/commons-discovery/

Commons-fileupload.jar :

Ce fichier permet de faciliter la mise en œuvre de fonctionnalités de type « upload de fichier » dans une webapp.

Sources : http://commons.apache.org/proper/commons-fileupload/

http://www.jmdoudoux.fr/java/dej/chap-apache_commons.htm

Commons-httpclient.jar :

Le paquet java.net permet à l'application d'avoir accès aux ressources via le http. Cependant, il y a des problèmes au niveau de la flexibilité et des fonctionnalités, d'où l'utilisation de l'archive Commons-httpclient.jar. Ce dernier fournis un paquet plus efficace, plus complet et à jour.

Source : http://hc.apache.org/httpcomponents-client-ga/

Commons-IO.jar :

Ce fichier fournit des utilitaires pour les opérations de type entrée/sortie. On en compte 6 principales :

- Les classes utilitaires : contient les méthodes permettant d'exécuter des tâches dites « communes »,
- Input : Flux d'entrée,
- Output : Flux de sortie,
- Filtre : Filtre les fichiers,
- Comparateur : Application de java.utils.comparator pour les fichiers,
- Moniteur de fichiers : Permet de contrôler les évènements de système de fichiers.

Source : http://commons.apache.org/proper/commons-io/

Commons-lang.jar :

Les bibliothèques java standard ne fournissent pas assez de méthodes permettant la manipulation de ses classes principales. Commons-lang.jar fournit ces classes manquantes (méthodes de manipulation, Réflexion d'objet -> voir Commons-beanutils.jar, …) ainsi que des améliorations d'aide à la construction de méthodes (toString, hashCode par exemple) et des améliorations de java.util.Date. Commons-lang.jar permet donc de fournir des utilitaires de base.

Source : http://commons.apache.org/proper/commons-lang/

Commons-logging.jar :

Commons-logging.jar est un Wrapper (patron de conception convertissant l'interface d'une classe en une interface pour le client) permettant l'utilisation de plusieurs implémentations d'API de logging (traitement d'application permettant l'émission et le stockage de messages suite à des évènements liés à l'exécution/l'utilisation de cette dernière).

Source : http://www.jmdoudoux.fr/java/dej/chap-apache_commons.htm

Displaytag.jar:

Cette librairie permet de traiter une liste d'objets (affichage de colonnes, design de la table en XHTML, …). Elle fournit donc des patrons de présentation web, compatibles avec le modèle MVC.

Source : http://www.displaytag.org/10/index.html

Displaytag-export-poi.jar:

Ce fichier est un module facultatif de displaytag.jar. Il permet d'exporter des données utilisant le format binaire de Microsoft Excel avec Jakarta POI.

Source : *http://mvnrepository.com/artifact/displaytag/displaytag-export-poi*

Dom4j.jar :

Dom4j est une API permettant la lecture, l'écriture (en XML) ainsi que la réalisation de requêtes Xpath(langage utilisé pour localiser une portion de document XML) sur des documents XML.

Source : *http://blog.paumard.org/cours/xml/chap04-dom-sax-utiliser-dom4j.html*

Freemarker.jar :

Freemarker est un outil générique permettant de produire/générer du texte en se basant sur des modèles (que ce dernier fournit) et des données (Java, XML).

Sources : *http://mvnrepository.com/artifact/org.freemarker/freemarker/2.3.20*

http://igm.univ-mlv.fr/~dr/XPOSE2005/freemarker/freemarker.php

IText.jar :

IText fournit une interface de programmation permettant la manipulation et la création de documents PDF.

Source : *http://itextpdf.com/products*

https://fr.wikipedia.org/wiki/IText

Log4j.jar :

Cette API permet de paramétrer et d'utiliser les logs (traitement d'application permettant l'émission et le stockage de messages suite à des évènements liés à l'exécution/l'utilisation de cette dernière).

Source : *http://www.jmdoudoux.fr/java/dej/chap-logging.htm#logging-2*

Mail.jar/Mailapi.jar :

Mail.jar/Mailapi.jar ou JavaMail est une API permettant l'utilisation d'email dans une application Java.

Source : *http://www.jmdoudoux.fr/java/dej/chap-javamail.htm#javamail-1*

Ognl.jar :

Ognl (Object-Graph Navigation Language) est un « langage d'expression » permettant d'interagir avec les objets ou les instances Java. En effet, il permet d'accéder aux propriétés/attributs des objets et des méthodes et de les appelés. Il permet aussi de projeter des listes, des sélections et autres expressions lambda.

Source : *http://commons.apache.org/proper/commons-ognl/*

Saaj.jar :

L'API SAAJ (SOAP with Attachments API for Java) permet d'effectuer des tâches de bas niveau sur les services web comme manipuler l'entête, … . Il offre la possibilité (standard) d'envoyer des documents XML depuis la plateforme Java.

Sources : *http://www.jguru.com/*

http://www.objis.com/formation-java/tutoriel-webservice-creation-web-service-java-6-jax-ws.html

Struts2-core.jar :

Struts2-core est un Framework utile au développement d'application web. En effet, il permet aux développeurs/designers de mieux répartir les tâches et ainsi pouvoir mieux se consacrer/gérer leur partie du projet (architecture MVC-Modèle Vue Controlleur). De plus, Struts2 permet d'automatiser la gestion de certains aspects comme la validation des données entrées par les utilisateurs via l'interface d'application.

Source : *https://fr.wikipedia.org/wiki/Apache_Struts*

Wsdl4j.jar:

Wsdl4j (Web Services Description Language for Java Toolkit) permet de créer, d'afficher et de manipuler des documents WSDL.

Sources : *http://jm.doudoux.pagesperso-orange.fr/java/tutorial/chap044.htm#chap_44_3*

http://sourceforge.net/projects/wsdl4j/

Xmlsec.jar :

XML Security fournit des normes de sécurité pour XML.

Source : *http://mvnrepository.com/artifact/xml-security/xmlsec/1.3.0*

Xwork-core.jar :

Xwork permet, sur certaines URLs, le déclenchement de l'exécution du code en modèle MVC.

Source : *https://docs-old.servicerocket.com/display/ATLASSIAN/Plugin+Example+-+XWork+Action*

2.2.2. Développement : Ajout du critère arrondissement

2.2.2.1. Modification de Result.jsp

- Label

Ajout d'un nouveau texte, « Arrondissement », précédant le champ de saisie. Pour cela, la balise « Text » récupère le texte à afficher dans « ResultAction.properties ».

> *Affichage du Label*

Ajout de la ligne de code suivante : « resultAction.arrondissement= « Arrondissement : » » dans ResultAction.properties.

```
#JSP Messages
resultAction.title = Popup Base d''adresses - Recherche d''adresses
resultAction.resultSearch = Résultat de la recherche
resultAction.adresse = Adresse :
resultAction.arrondissement = Arrondissement :
resultAction.titre = Recherche d''adresses
```

- Textfield

Ajout du champ de saisie/zone de texte du critère Arrondissement. Ce « textfield » récupèrera l'information saisie par l'utilisateur et l'affectera à la variable 'arrondissement' (name= « arrondissement »).

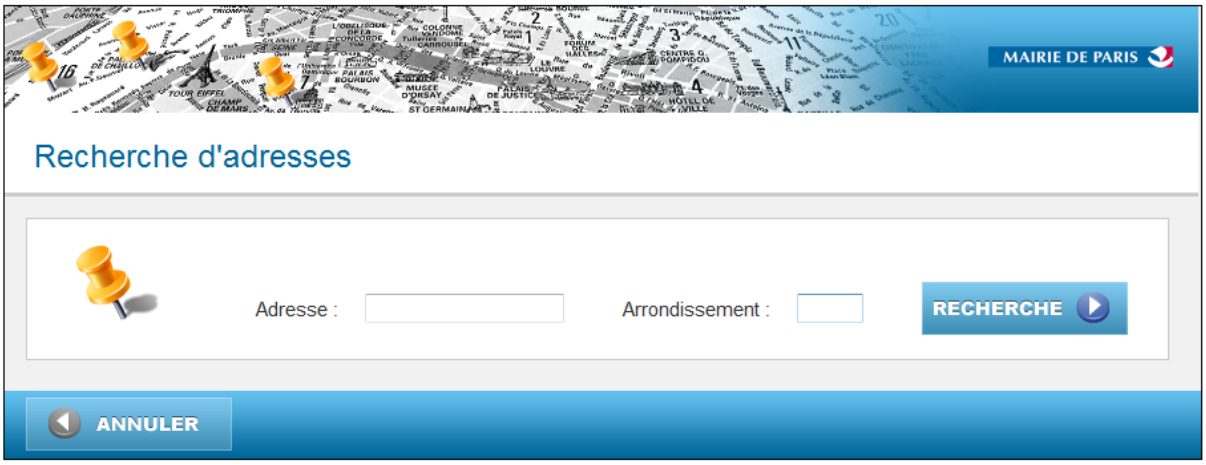
```
<s:text name="resultAction.arrondissement" />
<s:textfield name="arrondissement" />

<s:submit action="SearchAction" value="" onclick="recherche()"
    cssClass="bouton_recherche" />
```

- Fichier CSS « style-bleu »

Ajout d'une classe css nommée « inputArrondissement », gérant la partie graphique du critère texte de l'arrondissement affiché sur la popup.

```
.inputArrondissement{
    margin-left: 16px;
    margin-right: 40px;
    vertical-align: middle;
    width: 50px;
    boder: 1px solid #CCC;
}
```



2.2.2.2. Modification de PopupBAConstant

- Nouveau paramètre

Ajout d'une nouvelle constante PARAMETER_BA_CONSTANT contenant la variable 'arrondissement' qui stocke l'information saisie par l'utilisateur. Ce paramètre est en « public static final » (accessible à toutes les classes de l'application mais non modifiable).

```
/**
 * Constantes paramètres
 */
public static final String PARAMETER_BA_ADR_LIBRE = "BA_ADR_LIBRE";
public static final String PARAMETER_BA_CSS = "BA_CSS";
public static final String PARAMETER_BA_ADR_FOURNIE = "BA_ADR_FOURNIE";
public static final String PARAMETER_BA_NOM_FONCTION = "BA_NOM_FONCTION";
public static final String PARAMETER_BA_ADR_INCONNUE = "BA_ADR_INCONNUE";
public static final String PARAMETER_BA_USER_LOGIN = "BA_USER_LOGIN";
public static final String PARAMETER_BA_USER_PASS = "BA_USER_PASS";

public static final String PARAMETER_BA_ARRONDISSEMENT= "arrondissement";
```

- Accesseur

Seul l'ajout du getter correspondant à cette constante est nécessaire.

```
/**
 * @return the parameterBaArrondissement
 */
public String getParameterBaArrondissement() {
    return PARAMETER_BA_ARRONDISSEMENT;
}
```

2.2.2.3. Modification de ResultAction.java

- Nouvelle variable

Initialisation de la variable 'arrondissement' dans les propriétés et ajout des getters/setters correspondants.

```
//Paramètres
private String adresse_libre = "";
private String BA_CSS =  "../" + PropertiesUtils.getProperty(PopupBAConstant.DEFAULT_CSS);
private String adresse_fournie = "";
private String nom_fonction = "";
private String adresse_inconnue = "";
private String adresse;
private String user_login = PropertiesUtils.getProperty(PopupBAConstant.WS_BA_LOGIN);
private String user_pass = PropertiesUtils.getProperty(PopupBAConstant.WS_BA_MDP);

private String arrondissement;
```

- Affectation de la constante de saisie

Ajout d'une nouvelle condition (boucle) permettant de stocker la constante PARAMETER_BA_CONSTANT (importée de PopupBAConstant.java) dans la variable 'arrondissement' initialisée précédemment. Cette condition a été ajoutée à la fonction « execute() ». La propriété 'arrondissement' a également été ajoutée aux paramètres de « creationList ».

```
if(request.getParameter(PopupBAConstant.PARAMETER_BA_ARRONDISSEMENT) != null )
{
    arrondissement = request.getParameter(PopupBAConstant.PARAMETER_BA_ARRONDISSEMENT);
}
```

```
if(!adresse_libre.equals(""))
{
    /*lancement de la recherche avec récupération du code retour utilisé pour les traitements de la page*/
    logger.info(getText("resultAction.run"));
    resultat    = popupBaseAdressesService.creationList(adresse_libre, arrondissement, message);
```

2.2.2.4. Modification de PopupBaseAdressesService.java

- Nouvelle fonction

Ajout d'une nouvelle fonction « conversion() » permettant de convertir le numéro de l'arrondissement saisi par l'utilisateur en code Insee.

Pour cela, l'arrondissement saisi est converti en entier. Cela permet, en plus, de supprimer les zéros superficiels pouvant être placés par l'utilisateur avant le numéro de l'arrondissement, ce qui pourrait fausser la recherche.

```
// fonction permettant de convertir le numéro de l'arrondisser
public String conversion(String arrondissement){
    if (arrondissement!=null && !arrondissement.equals("")){
        //On converti l'arrondissement en entier pour lancer ;
        int value=Integer.parseInt(arrondissement);

        if (value>=1 && value <=9){
            return "7510"+value;
        }
        else if (value >=10 && value<=20){
            return "751"+value;
        }
        //si l'arrondissement n'existe pas, il sera par défau
        else{
            return "0";
        }
    }
    return "";
}
```

- Modification de la fonction creationList()

> Utilisation de conversion()

Tout d'abord, on contrôle la valeur saisie par l'utilisateur dans 'arrondissement' :

- Suppression de tous les caractères qui ne sont pas des chiffres.

- Vérification de la taille de 'arrondissement' : si la variable ne contient plus aucun caractère après la suppression, le critère arrondissement ne sera pas pris en compte lors de la recherche.

Si, après le contrôle, 'arrondissement' n'est pas vide, on appelle la fonction « conversion() ». Le résultat (donc le code Insee correspondant à l'arrondissement saisi dans la popup) est récupéré par la variable locale 'codeI'.

Une autre variable locale, 'codeI2', est également initialisée. Elle stockera la valeur de 'codeI' (qui est sous forme de chaine de caractère mais converti en entier). Cela permettra plus tard de vérifier si la valeur saisie dans arrondissement est valide.

```
//codeI récuperera le code insee sous forme de chaine de caractere
String codeI=null;

//codeI2 permettra de verifier si la valeur saisie dans arrondisseme
int codeI2=0;

//on supprime tout les caractères saisi par l'utilisateur qui ne son
arrondissement=arrondissement.replaceAll("\\D+","");
logger.info("DEBUG : arrondissement.replaceAll : "+ arrondissement);

//on vérifie qu'"arrondissement" n'est pas vide après suppression de
if (arrondissement.length()!=0){
    if (!arrondissement.trim().equals("")){
        codeI=conversion(arrondissement);
        logger.info("DEBUG : codeI : "+ codeI);
        codeI2=Integer.parseInt(codeI);
    }
}
```

> *Ajout du critère code Insee lors de la recherche*

- *Utilisation du service web*

Pour la recherche, deux fonctions distantes peuvent être utilisées : « searchAddress() » et « searchAddress_v2() ». Ces deux fonctions renvoient la liste d'adresses correspondant à la saisie de l'utilisateur.

Afin de déterminer laquelle sera appelée, des conditions ont été ajoutées :

- Si 'codeI' n'est pas null et que codeI2 (donc l'arrondissement converti en code Insee et traduit en entier) est supérieur ou égal à 75101 mais inférieur ou égal à 75120 :

```
/* appel de la fonction distante searchAddress_v2 (si l'utilisateur a renseigné le champ "Arrondissement") qui renvoie une
 if (codeI!=null && !codeI.trim().equals("") && codeI2>=75101 && codeI2<=75120){
     responseWebService = ((AdresseServicePortType) portType ).searchAddress_v2(zone, adresse, codeI, null, null, "apur");

     if(responseWebService == null)
     {
         logger.info( "--> L'adresse saisie est inconnue." );
         message.append("L'adresse saisie est inconnue");
         return NO_RESULT_FOUND;

     }

     /* traitement sur le flux contenu */
     makeAddressListInsee(responseWebService,codeI,message);
     }
```

« searchAddress_v2()» sera appelée. Elle prend en entrée (comme critère de recherche) la zone (ville), l'adresse saisie, la sortie (identifiant) et la date ainsi que le code Insee et le producteur. La sortie et la date sont, par défaut, déclaré 'null' et le producteur à 'apur'.

```
/**
 * Recherche d'une adresse avec producteur (retourne XML des attributs)
 */
public java.lang.String searchAddress_v2(java.lang.String zone, java.lang.String postalAddress, java.lang.String codeInsee, java.lang.String srs, java.lang.String date, java.lang.String productor) throws java
```

- Si 'codeI' est null :

```
/* appel de la fonction distante searchAddress (si l'utilisateur n'a pas renseigné le champ "Arrondisse
else if (codeI==null){
    responseWebService = ((AdresseServicePortType) portType ).searchAddress(zone, adresse, null, null);

    if(responseWebService == null)
    {
        logger.info( "--> L'adresse saisie est inconnue." );
        message.append("L'adresse saisie est inconnue");
        return NO_RESULT_FOUND;

    }

    /* traitement sur le flux contenu */
    makeAddressList(responseWebService);
}
```

« searchAddress()» sera appelée. Elle prend en entrée (comme critère de recherche) la zone (ville), l'adresse saisie, la sortie (identifiant) et la date. La sortie et la date sont, par défaut, déclarées 'null'.

```
/**
 * Recherche d'une adresse (retourne XML des attributs)
 */
public java.lang.String searchAddress(java.lang.String zone, java.lang.String postalAddress, java.lang.String srs, java.lang.String date) throws java.rmi.RemoteException
```

- Si codeI2 est égal à 0 (donc que l'arrondissement saisi n'est pas valide -> voir fonction conversion) :

On retourne « NO_RESULT_FOUND » ainsi que le message suivant : "L'arrondissement saisi est incorrect"

```
/*Message d'erreur lorsque l'arrondissement saisi est incorrect*/
else if (codeI2==0){
    message.append("L'arrondissement saisi est incorrect");
    return NO_RESULT_FOUND;
}
```

- *Création et traitement de la liste*

En fonction de la méthode distante appelée, deux fonctions peuvent être utilisée pour la création de la liste d'adresse :

- Si « searchAdresse() » est utilisée, ce sera la méthode « makeAddressList() » qui sera appelée.

```
/* traitement sur le flux contenu */
makeAddressList(responseWebService);
```

- Si « searchAdresse_v2() » est utilisée, ce sera la méthode « makeAddressListInsee() » qui sera appelée. Contrairement à « makeAddressList() », cette fonction contient une condition permettant de « trier » la liste d'adresses afin de ne garder que celles dont le code Insee des adresses retournées par le service web correspond au code Insee saisi par l'utilisateur.

```
/* traitement sur le flux contenu */
makeAddressListInsee(responseWebService,codeI,message);
```

```
if(myInseeCode!= null && myAddress.getCode_insee()!= null && myInseeCode.equalsIgnoreCase(myAddress.getCode_insee().trim())){
    addressList.add(myAddress);

}
```

- Si aucune des adresses retournées par le service web ne possède de code Insee identique au code Insee retourné par la conversion, une condition ajoutée à la méthode « creationList() » retournera « NO_RESULT_FOUND » et affichera le message suivant : « Cette adresse n'existe pas dans l'arrondissement saisi » .

```
/*Message d'erreur lorsque aucune adresse n'existe dans l'arrondissement saisi*/
if(addressList != null && addressList.size()== 0){
    message.append("Cette adresse n'existe pas dans l'arrondissement saisi");
    return NO_RESULT_FOUND;
}
```

2.2.2.5. Conclusion

Les modifications apportées à l'application permettent :

- L'ajout d'un nouveau champ « Arrondissement » à la popup
- La récupération du numéro de l'arrondissement saisi dans la popup
- La conversion du numéro de l'arrondissement saisi en code Insee
- Le contrôle et le traitement de cette saisie
- L'envoi de ce critère au service web via la fonction distante « searchAddress_v2() »
- Le retour, en XML, d'une liste d'adresse prenant en compte l'arrondissement saisi à partir du code Insee.
- La gestion des erreurs (saisie incorrect,…)

3. Situation Professionnelle – Rédaction d'une spécification fonctionnelle

3.1. Description et analyse de la situation professionnelle

3.1.1. Le contexte

L'application de recherche a été modifiée. Elle est maintenant opérationnelle mais ne remplit pas encore toutes les conditions requises pour être mise en production.

3.1.2. L'existant

Dans ce projet, la pop-up précédemment modifiée devra être étudiée.

3.1.3. Le besoin

L'application modifiée doit être accompagnée d'une spécification fonctionnelle, c'est-à-dire d'un cahier des charges du produit fini, couvrant l'intégralité des cas d'utilisation du produit en expliquant ce qu'il doit faire.

3.1.4. Démarche de résolution

Les différentes étapes de ce projet furent réalisées dans l'ordre suivant :

- Documentation sur la structure d'une spécification fonctionnelle,

- Documentation sur la réalisation d'un diagramme BPM,

- Rédaction de la spécification fonctionnelle.

3.2. Production

3.2.1. Spécification fonctionnelle de l'application de sélection d'adresse avec le critère arrondissement

La spécification fonctionnelle complète sera disponible en annexe.

3.2.1.1. Présentation de la solution

La solution présentée est une pop-up de sélection d'adresses. Cette dernière est ouverte sur le navigateur par une fonction Javascript qui prend en paramètre une adresse et un arrondissement, saisis par l'utilisateur. La pop-up sollicite ensuite une application J2EE autonome de sélection.
Cette application :

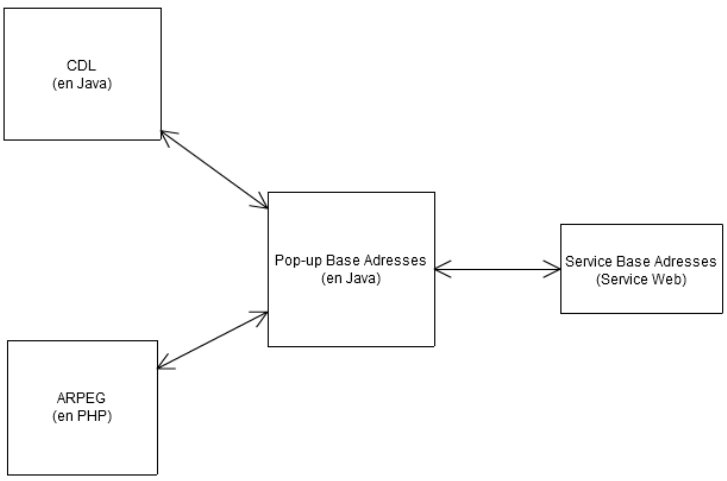
Convertit, par une fonction Javascript, l'arrondissement saisi en code Insee.

Interroge, en mode configurable http ou https, le service d'adresses de Base Adresses en transmettant l'adresse saisie par l'utilisateur et le code Insee précédemment converti.

Contrôle et trie la liste d'adresses retournées par Base Adresses, puis construit un formulaire de sélection d'adresses.

Une fois l'adresse sélectionnée, transmet à la fenêtre principale, par une autre fonction Javascript, les informations relatives à l'adresse sélectionnée par l'utilisateur.

3.2.1.2. Schéma général



3.2.1.3. Les fonctions distantes

Dans cette solution, plusieurs appels à Base Adresse sont effectués :
- searchAddress()
- searchAddress_v2()
- getAddressInfo()

La méthode searchAddress() renvoie une liste d'adresses contenant une partie des informations souhaitées. Elle prend comme critère de recherche l'adresse saisie par l'utilisateur (entre autres).

La méthode searchAddress_v2() renvoie une liste d'adresses contenant une partie des informations souhaitées. Elle prend comme critère de recherche l'adresse saisie par l'utilisateur ainsi que le code Insee (entre autres).

La méthode getAddressInfo() permet, pour chaque adresse, de récupérer les informations manquantes.

3.2.1.4. Nouveau paramètre

PARAMETER_BA_ARRONDISSEMENT : Arrondissement saisi par l'utilisateur (sous forme d'entier entre 1 et 20).

3.2.2. Spécification fonctionnelle détaillée du projet

3.2.2.1. Scénario de recherche nominal

1 - execute()
Récupération des paramètres saisis par l'utilisateur et appel du constructeur PopupBaseAdressesService().

2 - PopupBaseAdressesService()
Initialisation de la connexion avec le service web Base Adresses.

3 - execute()
Appel de la méthode creationList() en lui retournant les paramètres saisis.

4 - creationList()
Contrôle des informations saisies par l'utilisateur puis appel de la fonction conversion.

5 - conversion()

Contrôle de l'entier (il faut qu'il soit compris entre 1 et 20) et conversion de l'arrondissement en code Insee.

6 - creationList()
Appel de la fonction searchAddress() ou searchAddress_v2().

5 - searchAddress()
Interrogation du service d'adresses de Base Adresses par une requête http (ou https en fonction de la configuration) en passant en paramètre l'adresse complète sous la forme attendue par Base Adresses.
Cette fonction est appelée si aucun arrondissement n'a été saisi ou si ce dernier n'est pas valide.

6 - searchAddress_v2()
Interrogation du service d'adresses de Base Adresses par une requête http (ou https en fonction de la configuration) en passant en paramètre l'adresse complète ainsi que le code Insee sous la forme attendue par Base Adresses.

7 - creationList()
Récupération de la réponse du service Base Adresses puis appel de la fonction makeAddressListInsee() ou makeAddressList() auxquels seront retourné cette réponse.

8 - makeAddressList()
Structuration de la réponse du service Base Adresse. Pour cela, makeAddressList() appel la fonction distante getAddressInfo(), puis renvoi la liste d'adresses complète qui sera affichée par la pop-up.
Elle n'est appelée que si searchAddress() est utilisée.

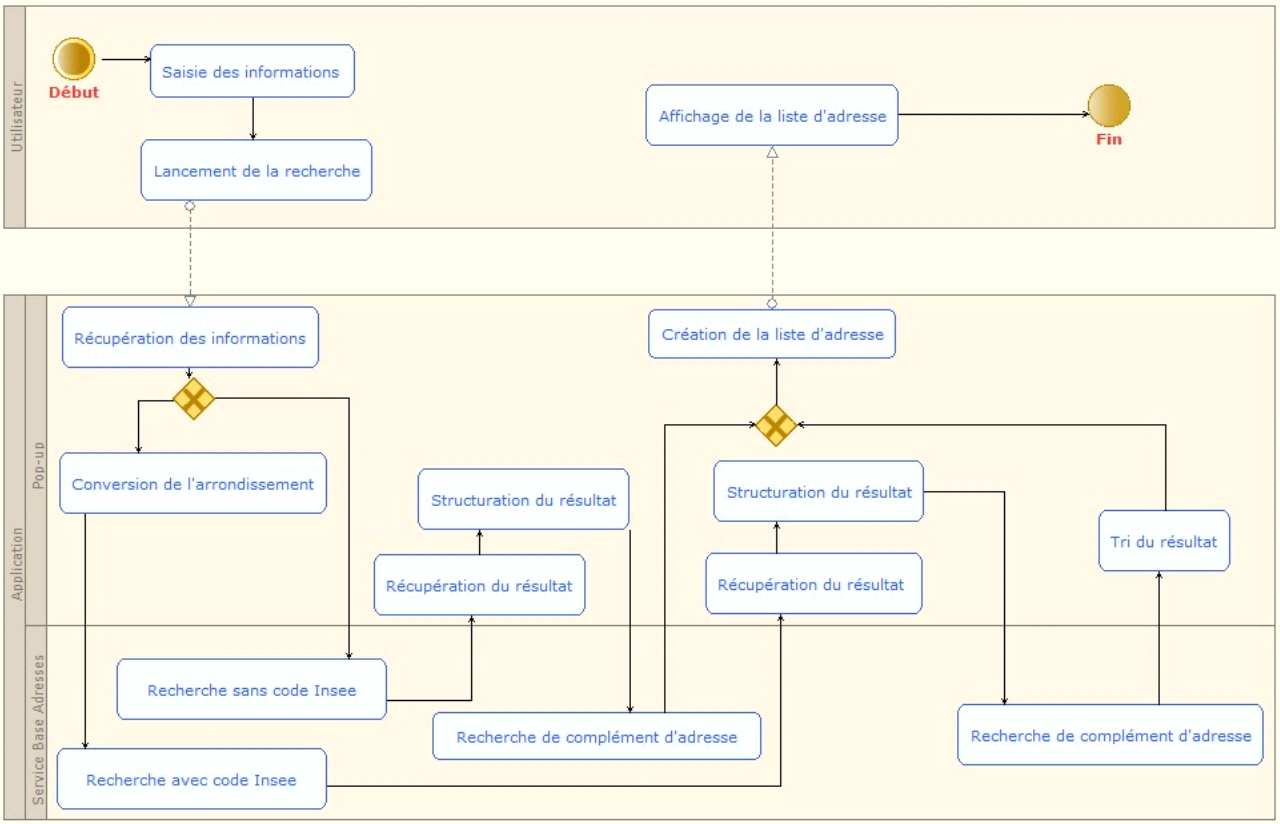
9 - makeAddressListInsee()
Structuration de la réponse du service Base Adresse. Pour cela, makeAddressListInsee() appelle la fonction distante getAddressInfo(), puis renvoie la liste d'adresses complète qui sera affichée par la pop-up.
makeAddressListInsee() trie également les adresses renvoyées par le service web : seules les adresses dont le code Insee correspond au code Insee retourné par conversion() seront conservées.
Elle n'est appelée que si searchAddress_v2() est utilisée.

10 - getAddressInfo()
Pour chaque adresse retournée par Base Adresses, l'application de sélection interroge le service d'adresses de Base Adresses en passant en paramètre l'identifiant de l'adresse. Base Adresse retourne ensuite les informations complètes de l'adresse dans addresseInfo.

11 - execute()
Appel de la fonction getAddressList().

12 - getAddressList()
Retourne la liste d'adresses précédemment traitée qui sera affichée par la pop-up.

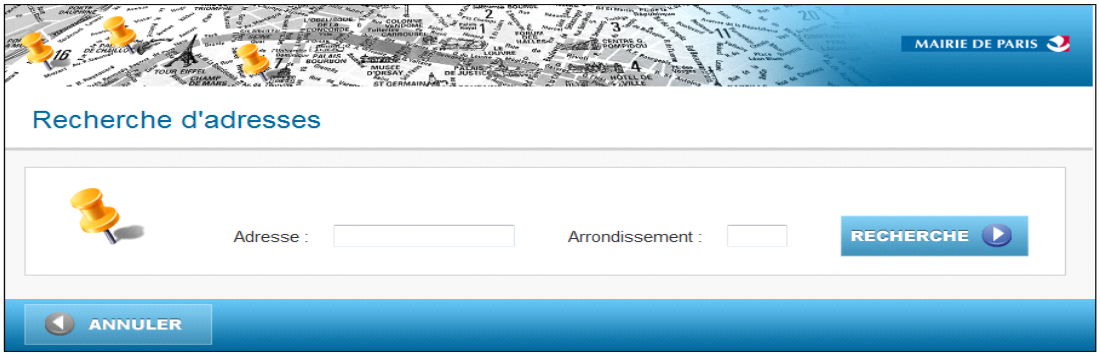
Le scénario peut également être traduit par le diagramme BPM suivant :



3.2.2.2. Scénario de recherche en cas de succès

A1 - Aucuns des champs ne sont renseigné par l'utilisateur :

Aucune recherche n'est lancée, la pop-up n'affiche rien sur son interface (pas de message d'erreur).



A2 - Le champ adresse est renseigné, mais pas l'arrondissement :

La recherche est effectuée, le critère 'arrondissement' n'est pas pris en compte (mis par défaut).

Cet enchainement démarre au point 4 du scénario nominal :

4 - creationList()
Contrôle des informations saisies par l'utilisateur et appel de la fonction searchAddress().

5 - searchAddress()
Interrogation du service d'adresses de Base Adresses par une requête http (ou https en fonction de la configuration) en passant en paramètre l'adresse complète sous la forme attendue par Base Adresses.

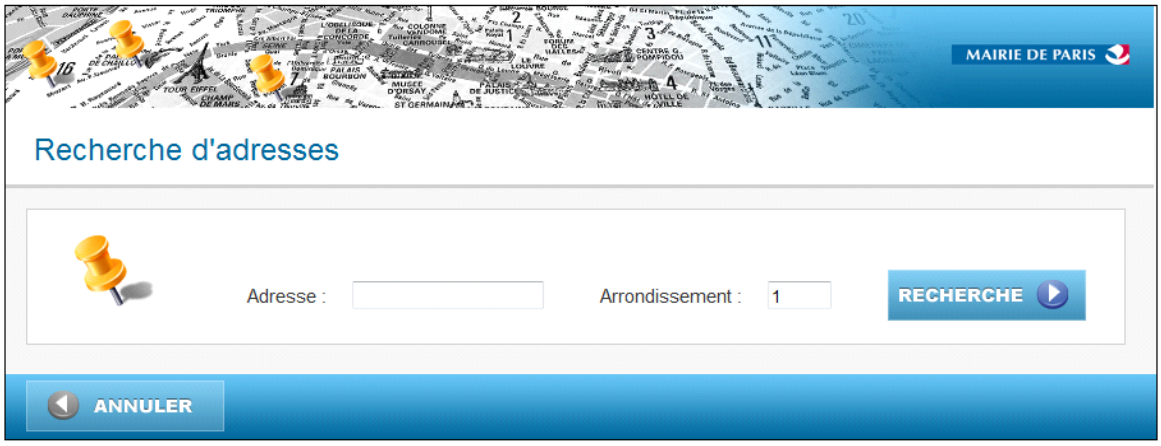
6 - creationList()
Récupération de la réponse du service Base Adresses puis appel de la fonction makeAddressList() à laquelle sera retournée cette réponse.

7 - makeAddressList()
Structuration de la réponse du service Base Adresse. Pour cela, makeAddressList() appelle la fonction distante getAddressInfo(), puis renvoie la liste d'adresses complète qui sera affichée par la pop-up.

Le déroulement reprend au point 10 du scénario nominal.

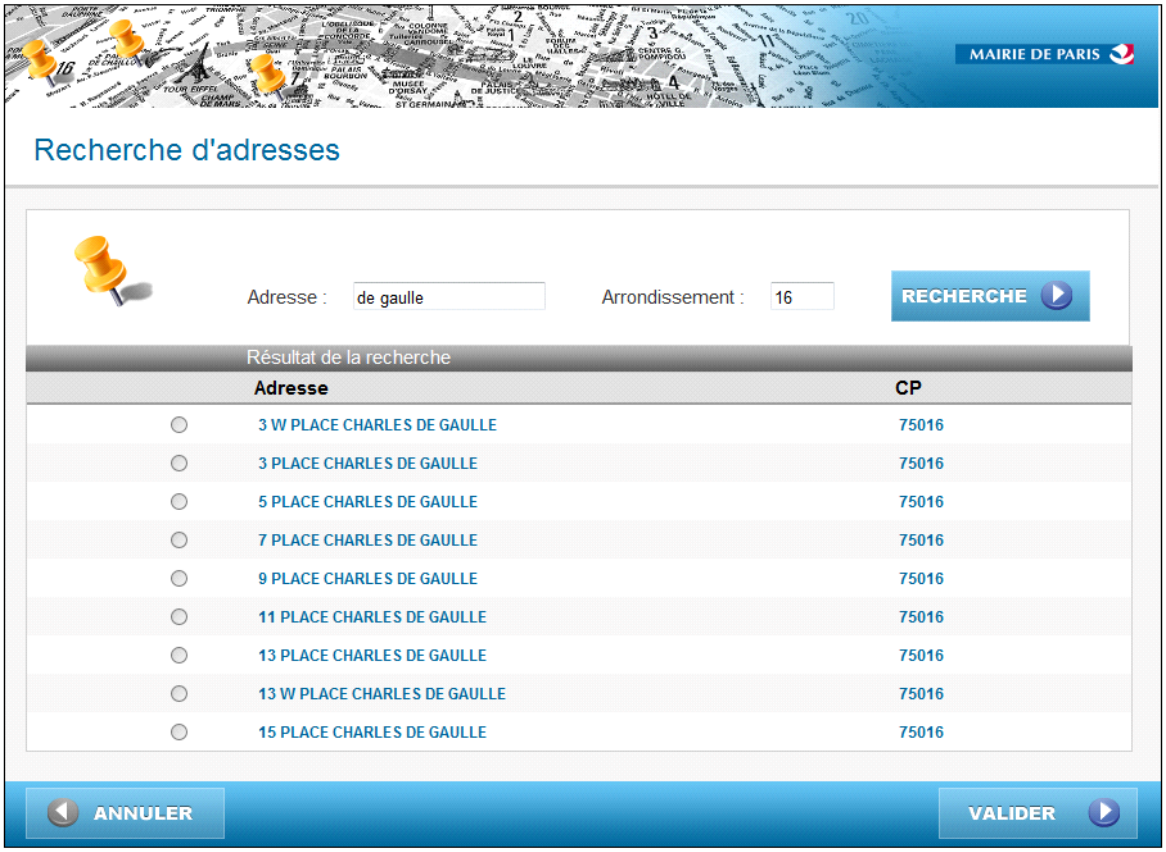
A3 - Le champ arrondissement est renseigné, mais pas celui de l'adresse :

Aucune recherche n'est lancée, la popup n'affiche rien sur son interface (pas de message d'erreur).



A4 - Les champs Adresse et Arrondissement sont renseignés :

La recherche est lancée, le service web renvoie une liste d'adresses correspondant à la saisie.



Cet enchainement démarre au point 6 du scénario nominal :

6 - creationList()
Appel de la fonction searchAddress_v2().

7 - searchAddress_v2()
Interrogation du service d'adresses de Base Adresses par une requête http (ou https en fonction de la configuration) en passant en paramètre l'adresse complète ainsi que le code Insee sous la forme attendue par Base Adresses.

8 - creationList()
Récupération de la réponse du service Base Adresses puis appel de la fonction makeAddressListInsee() à laquelle sera retournée cette réponse.

9 - makeAddressListInsee()
Structuration de la réponse du service Base Adresse. Pour cela, makeAddressListInsee() appelle la fonction distante getAddressInfo(), puis renvoie la liste d'adresses complète qui sera affichée par la pop-up.
makeAddressListInsee() trie également les adresses renvoyées par le service web : seules les adresses dont le code Insee correspond au code Insee retourné par conversion() seront conservées.

Le déroulement reprend au point 10 du scénario nominal.

A5 - L'arrondissement est valide mais ne comporte pas que des entiers

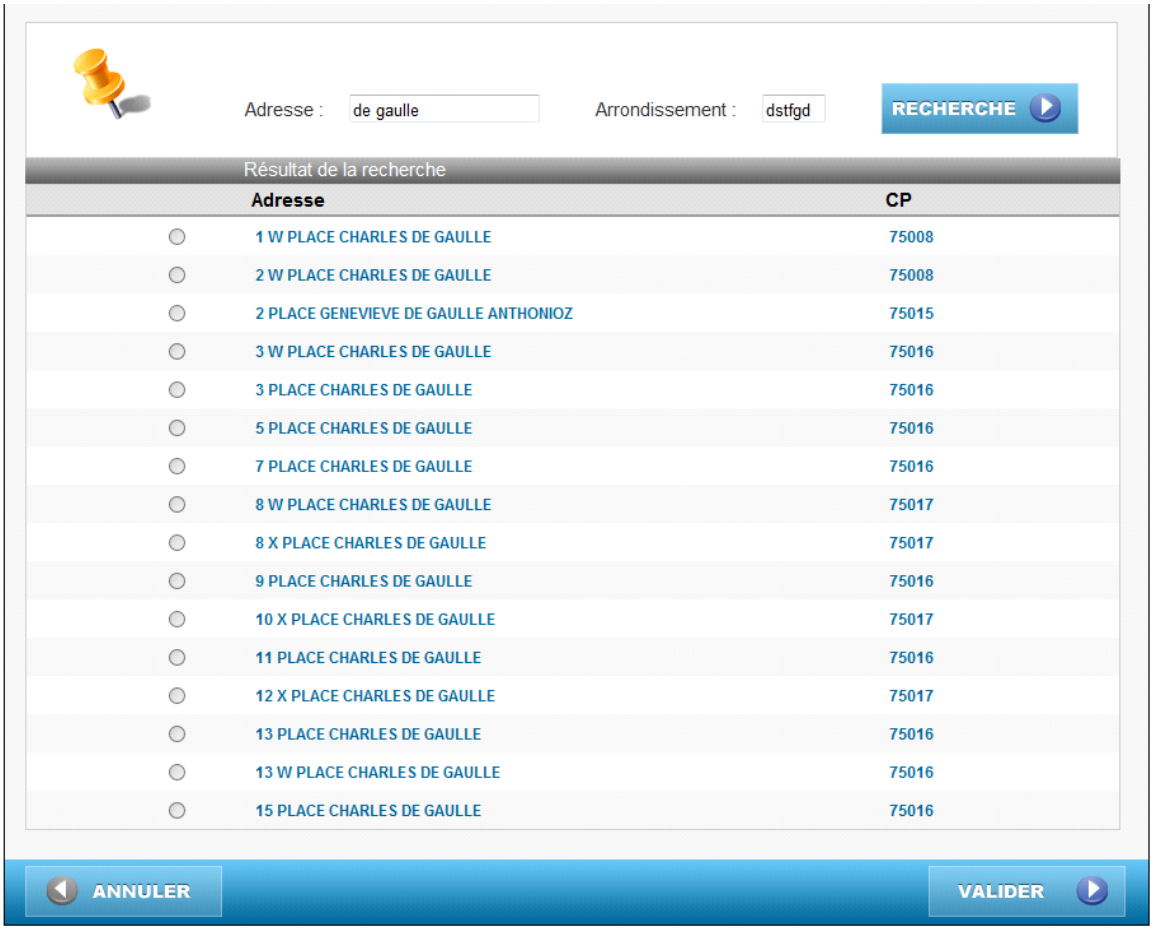
La recherche est lancée, le service web renvoi une liste d'adresses correspondant à la saisie.

L'enchainement du scénario B1 reprend donc celui du scénario A4 car les caractères non numériques sont supprimés lors du contrôle de creationList() (voir point 4 du scénario nominal).

3.2.2.3. Scénario de recherche en cas d'échec

B1 - Le champ arrondissement ne contient aucun entier :

Le critère arrondissement n'est pas pris en compte, et la recherche se fait uniquement en fonction du critère 'adresse'.



L'enchainement du scénario B1 reprend donc celui du scénario A2 car les caractères non numériques sont supprimés lors du contrôle de creationList() (voir point 4 du scénario nominal).

B2 - L'arrondissement saisi par l'utilisateur n'est pas valide :

La pop-up affiche le message d'erreur « L'arrondissement saisi est incorrect ».

L'erreur est traitée à partir du point 5 du scénario nominal :

5 - conversion()
Contrôle de l'entier (il faut qu'il soit compris entre 1 et 20) et conversion de l'arrondissement en code Insee. L'arrondissement saisi étant 22, il sera mis par défaut à 0.

6 - creationList()
L'arrondissement étant défini par défaut à 0, le message d'erreur « L'arrondissement saisi est incorrect » est renvoyé à la pop-up et la fonction searchAddress_v2() ne peut être appelé. Le scénario est terminé.

B3 - L'arrondissement saisi ne correspond pas à l'adresse :

La pop-up affiche le message d'erreur «Cette adresse n'existe pas dans l'arrondissement saisi ».

L'erreur est traitée au point 9 du scénario nominal :

9 - makeAddressListInsee()
Après le tri il ne reste plus aucune adresse.

10 - creationList()
Le message d'erreur «Cette adresse n'existe pas dans l'arrondissement saisi » est renvoyé à la pop-up. Le scénario est terminé.

4. Situation Professionnelle – Gestion de l'indisponibilité du service web de l'application de sélection d'adresse

4.1. Description et analyse de la situation professionnelle

4.1.1. Le contexte

L'application de sélection d'adresse est connectée à un service web qui permet, en fonction des informations qui lui sont envoyé, de retourner une liste d'adresses complètes.
L'application est par contre inutilisable si elle n'est pas connectée au service web.

4.1.2. L'existant

Dans ce projet, la pop-up modifiée lors de la première situation professionnelle devra être utilisée.

4.1.3. Le besoin

La pop-up doit pouvoir répondre au besoin de l'utilisateur et non être paralysé par l'indisponibilité du service web.

4.1.4. Démarche de résolution

Les différentes étapes de ce projet furent réalisées dans l'ordre suivant :

- Test et simulation d'indisponibilité,

- Ajout des lignes de code permettant de gérer cette indisponibilité,

- Test et simulation d'indisponibilité,

- Rédaction d'un compte-rendu fonctionnel et technique, résumant les modifications apportées et expliquant comment l'accès au service web a été rendu non bloquant.

4.2. Production

4.2.1. Développement : Gestion de l'indisponibilité du service web

4.2.1.1. Modification de PopupBaseAdressesService.java

> Nouvelle fonction

Ajout d'une nouvelle fonction « isAvailable() » permettant de vérifier que le service web est disponible. Pour cela, une recherche utilisant la fonction distante « searchAddress() » est lancée. Elle prend en paramètre une adresse valide de la ville de Paris (ici 55 Rue du Faubourg Saint-Honoré).

Si le service web est disponible, la recherche est effectuée et « isAvailable() » renvoi 'True'.

Sinon, « isAvailable() » renvoi 'False'.

```
public boolean isAvailable()
{
    try{
        ((AdresseServicePortType) portType ).searchAddress(zone, "55 Rue du Faubourg Saint-Honoré, 75008 Paris", null, null);
    }catch (InvalidParameterException e)
    {
        logger.info( "--> " + e.getFaultString());
        return false;
    }
    catch ( RemoteException e )
    {
        logger.error("--> "+e.getMessage(  ));

        return false;
    }

    return true;
}
```

> Modification de fonction

La fonction « conversion() » a été modifiée. Si le service Base Adresses est indisponible, et donc que la méthode « isAvailable() » retourne 'False', la fonction conversion() convertira l'arrondissement saisi en code postal. Pour cela, une condition a été ajoutée :

```
// fonction permettant de convertir le numéro de l'arrondissement saisi par l'utilisateur en code Insee
public String conversion(String arrondissement){

    if (isAvailable()){
        if (arrondissement!=null && !arrondissement.equals("")){
            //On converti l'arrondissement en entier pour lancer la conversion
            int value=Integer.parseInt(arrondissement);
            if (value>=1 && value <=9){
                return "7510"+arrondissement;
                }
            else if (value >=10 && value<=20){
                return "751"+arrondissement;
                }
            //si l'arrondissement n'existe pas, il sera par défaut mis à 0
            else{
                return "0";
                }
            }
        return "";
        }

    else{
        if (arrondissement!=null && !arrondissement.equals("")){
            //On converti l'arrondissement en entier pour lancer la conversion
            int value=Integer.parseInt(arrondissement);
            if (value>=1 && value <=9){
                return "7500"+arrondissement;
                }
            else if (value >=10 && value<=20){
                return "750"+arrondissement;
                }
            //si l'arrondissement n'existe pas, il sera par défaut mis à 0
            else{
                return "0";
                }
            }
        return "";
        }
    }
```

4.2.1.2. Modification de ResultAction.java

> Nouvelle condition

Ajout d'une nouvelle condition, utilisé à l'ouverture de la pop-up et avant le lancement d'une recherche. Cette dernière fait appel à la méthode « isAvailable()».

Si elle renvoi 'True', cela veut dire que la connexion au service web est correctement effectuée. La/les recherche(s) peuvent être lancées normalement.

Si elle renvoie 'False', cela veut dire que le service web est indisponible : le message « Le service Base Adresses est actuellement indisponible. Vous pouvez compléter votre saisie mais celle-ci ne sera pas contrôlée. » est alors affiché. L'adresse renseignée par l'utilisateur sera alors retournée, sans être modifiée, par la pop-up si ce dernier effectue tout de même une recherche. Si un arrondissement est saisi, il sera converti en code postal puis affiché avec l'adresse.

```
if(popupBaseAdressesService.isAvailable())
{
    if( !adresse_libre.equals(""))
    {
        /*lancement de la recherche avec récupération du code retour utilisé pour les traitements de la page*/
        logger.info(getText("resultAction.run"));
        resultat    = popupBaseAdressesService.creationList(adresse_libre,arrondissement, message);

        /* si le paramètre BA_ADR_INCONNUE est égal à oui, alors on ajout la chaine de caractère reçu à la liste de résultat */
        if(adresse_inconnue.equals(OUI) && !adresse_libre.equals(""))
        {
            popupBaseAdressesService.addToAddressList(adresse_libre, arrondissement);
            resultat = PopupBAConstant.SEARCH_RESULT_FOUND;
        }

        logger.info(getText("resultAction.finish"));


        lstResults = popupBaseAdressesService.getAddressList();
    }
}
else
{
    message.append("Le service Base Adresses est actuellement indisponible. Vous pouvez compléter votre saisie mais celle-ci ne
    if( !adresse_libre.equals(""))
    {
        logger.info(getText("resultAction.run"));

        //on supprime tout les caractères saisi par l'utilisateur qui ne sont pas des chiffres
        arrondissement=arrondissement.replaceAll("\\D+","");

        //On converti en code postal l'arrondissement saisi apres avoir supprimer tout les potentiels espaces
        String codePostal=popupBaseAdressesService.conversion(arrondissement.trim());

        popupBaseAdressesService.addToAddressList(adresse_libre, codePostal);
        resultat = PopupBAConstant.SEARCH_RESULT_FOUND;
    }
    logger.info(getText("resultAction.finish"));
    lstResults = popupBaseAdressesService.getAddressList();
}

return SUCCESS;
```

4.2.2. Compte-rendu général et détaillé

4.2.2.1. Description générale du projet

La modification apportée à la solution permet de continuer d'utiliser la pop-up Base Adresses dans le cas où le service web ne serait plus disponible.

L'utilisateur pourra donc renseigner les champs adresse et arrondissement et lancer une recherche. Ces derniers ne seront par contre pas contrôlés.

La liste de sélection d'adresses affichera alors l'adresse saisie ainsi que l'arrondissement, converti en code postal.

4.2.2.2. Description détaillée du projet

4.2.2.2.1. Nouvelle fonction et condition

- isAvailable() : Cette fonction permet de vérifier si le service Base Adresses est disponible ou non. (True : le service web est disponible, False : le service web est indisponible)

- Ajout d'une nouvelle condition, utilisé à l'ouverture de la pop-up et avant le lancement d'une recherche. Elle permet de prévenir l'utilisateur si le service Base Adresse est indisponible, et de lancer une recherche en conséquence.

4.2.2.2.2. Modification de fonction

La fonction « conversion() » a été modifiée. Si le service web n'est pas accessible, elle convertira l'arrondissement saisi en code postal et non en code Insee.

4.2.2.2.3. Simulation de l'indisponibilité

Pour rendre le service web indisponible, il faut rendre la connexion à ce dernier impossible.

Pour cela, il suffit de geler (en mettant en commentaire par exemple) les paramètres d'identification :

```
//Attribution des paramètres d'identification de connexion au service web
//portType.setUsername( _UserName );
//portType.setPassword( _Password );
```

4.2.2.2.4. Exemple

Après avoir rendu le service web indisponible, la pop-up affiche le message suivant lors de son exécution :

Si l'utilisateur saisi tout de même des informations et lance une recherche, la liste de sélection d'adresse n'affiche alors que l'adresse saisie et l'arrondissement converti en code postal :

Si l'utilisateur ne renseigne que l'adresse, seulement cette dernière sera affichée :

5. Situation professionnelle – Tests de l'application finale

5.1. Description et analyse de la situation professionnelle

5.1.1. Contexte

Les modifications de l'application de recherche d'adresse sont toutes achevées. Aucun test n'as par contre été réalisé afin de vérifier son bon fonctionnement.

5.1.2. L'existant

Dans ce projet, la version finale de la pop-up de recherche d'adresse devra être utilisée.

5.1.3. Le besoin

Mise en place de tests unitaires dans l'application finale. Un cahier de recette doit également être réalisé.

5.2. Production

5.2.1. Les Tests Unitaires

La majorité des modifications ayant été effectuée dans la classe « PopupBaseAdressesServiceTest.java », les tests se concentreront sur cette dernière.

> Pré-requis :

```
private static final Logger logger = Logger.getLogger(PopupBaseAdressesServiceTest.class);
PopupBaseAdressesService service =  null;
AdresseServicePortType adresseTest= null;
List<ResultAddress> addressList;
private StringBuffer message = new StringBuffer("");


public PopupBaseAdressesServiceTest() {
    // TODO Auto-generated constructor stub
    service = new PopupBaseAdressesService("admin", "admin");
}
```

5.2.1.1. Test de la fonction de conversion

Ce test couvre différents cas de figures :

- Le nombre à convertir est un entier situé entre 1 et 9 :

```
@Test
public void conversion()
{
    String rslt = service.conversion("1");
    assertTrue(rslt !=null && rslt.equals("75101"));
    logger.info("TEST CONVERSION 1e ARR = "+rslt +" : OK");
```

Le test est un succès si la fonction conversion renvoi « 75101 » lorsque l'entier « 1 » est saisi en entrée.

- Le nombre saisi est un entier situé entre 10 et 20 :

```
rslt = service.conversion("10");
assertTrue(rslt !=null && rslt.equals("75110"));
logger.info("TEST CONVERSION 10 ARR = "+rslt +" : OK");
```

Le test est un succès si la fonction conversion renvoi « 75110 » lorsque l'entier « 10 » est saisi en entrée.

- Le nombre saisi est précédé par des 0

```
rslt = service.conversion("01");
assertTrue(rslt !=null && rslt.equals("75101"));
logger.info("TEST CONVERSION 01 ARR = "+rslt +" : OK");
```

Le test est un succès si la fonction conversion renvoi « 75101 » lorsque l'entier « 01 » est saisi en entré.

```
rslt = service.conversion("0010");
assertTrue(rslt !=null && rslt.equals("75110"));
logger.info("TEST CONVERSION 010 ARR = "+rslt +" : OK");
```

Le test est un succès si la fonction conversion renvoi « 75110 » lorsque l'entier «0010 » est saisi en entré.

- Le nombre à convertir est un entier accompagné de caractère

```
rslt = service.conversion("1er");
assertTrue(rslt !=null && rslt.equals("75101"));
logger.info("TEST AVEC CARACTERE = "+rslt +" : OK");
```

Le test est un succès si la fonction conversion renvoi « 75101 » lorsque la chaine de caractères « 1er » est saisie en entrée.

5.2.1.2. Test des modifications apportées à la fonction creationList()

Ce test permet de vérifier que la méthode fonctionne correctement pour chaque résultat qu'elle peut retourner :

- RESULT_FOUND (0)

- NO_RESULT_FOUND (1)

- TOO_MUCH_RESULT (2)

- ERROR (3)

- EMPTY_PARAMETER (4)

- Il y a un résultat à la recherche (RESULT_FOUND)

> *Les critères adresse et arrondissement sont valides*

```
//Un résultat est retourné
String adresse="chatelet";
String arrondissement="1er";
int rslt = service.creationList(adresse, arrondissement, message);
assertTrue( rslt == PopupBaseAdressesService.RESULT_FOUND);
logger.info("TEST = "+arrondissement +" : OK");
```

Le test est un succès si la fonction creationList renvoi 0 (RESULT_FOUND).

> *Le critère adresse est valide mais le critère arrondissement ne contient aucun entier*

```
//Que des caracteres dans le parametre arrondissement
String adresse6="de gaulle";
String arrondissement6="zqerfs";
rslt = service.creationList(adresse6, arrondissement6, message);
assertTrue( rslt == PopupBaseAdressesService.RESULT_FOUND);
logger.info("TEST = "+arrondissement6 +" : OK");
```

Le test est un succès si la fonction creationList renvoie 0 (RESULT_FOUND).

> *Les critères adresse et arrondissement sont valides mais l'arrondissement contient des caractères autres que des entiers*

```
//caracteres et entiers dans le parametre arrondissement
String adresse7="de gaulle";
String arrondissement7="8zqerfs";
rslt = service.creationList(adresse7, arrondissement7, message);
assertTrue( rslt == PopupBaseAdressesService.RESULT_FOUND);
logger.info("TEST = "+arrondissement7 +" : OK");
```

Le test est un succès si la fonction creationList renvoie 0 (RESULT_FOUND).

- Il n'y a aucun résultat à la recherche (NO_RESULT_FOUND)

Le critère adresse contient une adresse inconnue et le critère arrondissement n'est pas renseigné :

```
//Aucun résultat trouvé
String adresse2="sdfgfcx";
String arrondissement2="";
rslt = service.creationList(adresse2, arrondissement2, message);
assertTrue( rslt == PopupBaseAdressesService.NO_RESULT_FOUND);
logger.info("TEST = "+arrondissement2 +" : OK");
```

Le test est un succès si la fonction creationList renvoi 1 (NO_RESULT_FOUND).

- Il y a trop de résultats (TOO_MUCH_RESULT)

Le critère adresse contient une chaine de caractères retournant un grand nombre d'adresses et le critère arrondissement n'est pas renseigné :

```
//Trop de résultats retournés
String adresse3="diderot";
String arrondissement3="";
rslt = service.creationList(adresse3, arrondissement3, message);
assertTrue( rslt == PopupBaseAdressesService.TOO_MUCH_RESULT);
logger.info("TEST = "+arrondissement3 +" : OK");
```

Le test est un succès si la fonction creationList renvoie 2 (TOO_MUCH_RESULT).

- Il y a une erreur lors de la recherche (ERROR)

Cette exception n'est jamais appelée car une autre remote exception est lancée avant cette dernière.

- Aucun paramètre saisi (EMPTY_PARAMETER)

Les critères adresse et arrondissement ont été laissés vides par l'utilisateur :

```
//Aucun parametres saisis
String adresse5="";
String arrondissement5="";
rslt = service.creationList(adresse5, arrondissement5, message);
assertTrue( rslt == PopupBaseAdressesService.EMPTY_PARAMETER);
logger.info("TEST = "+arrondissement5 +" : OK");
```

Le test est un succès si la fonction creationList renvoie 4 (EMPTY_PARAMETER).

5.2.1.3. Test de la fonction de liste d'adresses

La fonction makeAddressListInsee étant publique, on ne peut la tester directement.

Pour réaliser ce test, nous faisons donc appel tout d'abord à la méthode creationList (qui elle-même appel makeAddressListInsee) en lui rentrant des paramètres valides : l'adresse sera « 100 rue réaumur » et l'arrondissement « 2 ».

Le test sera alors un succès si le résultat remplit deux conditions :

- Le resultat renvoyé par creationList n'est pas null et est composé d'au moins un élément.

- Le code Insee du résultat correspond au code Insee attendu, soit 75102.

```
@Test
public void makeAddressListInsee(){
    //cette méthode etant privée, on ne peux la tester directement
    service.creationList("100 rue reaumur ", "2",message);
    List<ResultAddress> lstResults = service.getAddressList();
    assertTrue(lstResults !=  null && lstResults.size()>0);
    for(ResultAddress addr : lstResults)
    {
        System.out.println("ADDRESS : "+addr);
        assertTrue(addr.getCode_insee().equals("75102"));
    }
```

5.2.1.4. Test de la fonction de vérification de disponibilité du service web

Pour tester cette méthode, il suffit simplement de simuler une vérification. Si après appel, isAvailable renvoie true, alors le test est un succès.

```
@Test
public void isAvailable() throws InvalidParameterException, RemoteException{
    boolean test=service.isAvailable();
    assertTrue( test == true);
}
```

5.2.2. Cahier de recette

Suite aux tests précédemment réalisés, la mise en place d'un cahier de recettes est nécessaire pour vérifier que l'expression des besoins a bien été respectée.

Modification de la Pop-up Base Adresses

#	Ecran	Spécifications/Règles de gestion	Test à effectuer	Résultats attendus	Résultats obtenus	Statut	# Anomalie	Fait O2T
1.1		L'adresse et l'arrondissement ont été renseignés et sont valide	Lancement de la recherche	Affichage d'une liste d'adresses prenant en compte ces deux critères		Succès		
1.2		L'adresse et l'arrondissement ont été renseignés et sont valide mais des caractères qui ne sont pas des entiers ont été saisis dans l'arrondissement	Lancement de la recherche	Affichage d'une liste d'adresses prenant en compte ces deux critères. Les caractères saisis dans arrondissement qui ne sont pas des entiers sont ignorés		Succès		
1.3		L'adresse est renseignée et est valide mais l'arrondissement saisi ne contient aucun entier	Lancement de la recherche	Affichage d'une liste d'adresses ne prenant en compte que le critère adresse		Succès		
1.4		L'adresse est renseignée et est valide mais n'existe pas dans l'arrondissement saisi	Lancement de la recherche	Affichage du message d'erreur "Cette adresse n'existe pas dans l'arrondissement saisi"		Succès		
1.5		L'adresse est renseignée et est valide mais le champ arrondissement est laissé vide	Lancement de la recherche	Affichage d'une liste d'adresse ne prenant en compte que le critère adresse		Succès		
1.6		L'adresse est renseignée et est valide mais l'arrondissement saisi n'existe pas	Lancement de la recherche	Affichage du message d'erreur "L'arrondissement saisi est incorrect"		Succès		
1.7		L'adresse et l'arrondissement sont renseignés et sont valide mais un (ou plusieurs) 0 ont été saisi devant l'entier correspondant à l'arrondissement	Lancement de la recherche	Affichage d'une liste d'adresses prenant en compte ces deux critères. Les 0 saisis dans arrondissement sont ignorés.		Succès		

1.8		Le service web est indisponible	Lancement de la Pop-up	Affichage du message d'erreur "Le service Base Adresses est actuellement indisponible. Vous pouvez compléter votre saisie mais celle-ci ne sera pas contrôlée."		Succès
1.9		Le service web est indisponible: les champs adresse et arrondissement sont renseignés et l'arrondissement existe	Lancement de la recherche	Affichage de l'adresse saisi et du code postal correspondant à l'arrondissement saisi		Succès
2.1		Le service web est indisponible: les champs adresse et arrondissement sont renseignés mais l'arrondissement n'existe pas	Lancement de la recherche	Affichage de l'adresse saisi. Le code postal est mis par défaut à 0		Succès
2.2		Le service web est indisponible: le champs d'adresse est le seul à etre renseigné	Lancement de la recherche	Affichage de l'adresse saisi. Aucun code postal n'est affiché		Succès
2.3		Le service web est indisponible: L'adresse est renseignée et aucun entier n'est saisi dans le champ arrondissement	Lancement de la recherche	Affichage de l'adresse saisi. Aucun code postal n'est affiché		Succès

MAIRIE DE PARIS

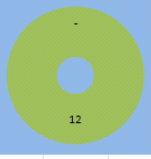
facil' familles
CENTRE DE COMPETENCES

Domaine :	FAC	Pop-up Base Adresses
Processus :	004	Modification

Reste à faire	-
Succès	12
Échec	-
Avancement :	100,00%

Commentaires :

Réinisitialisation du fichier : RAZ



Sous-processus		Reste à faire	Succès	Échec
Modification de la pop-up BA	Modification de la pop-up BA	0	12	0

Pour finir, le projet final a été livré au bureau des infrastructures et de la production qui sera chargé de l'installer sur les machine de prod (service d'études / réalisations et service d'exploitation chargé de garantir le bon fonctionnement sécurisé du service sur les serveurs dédiés aux usagers internes comme l'administration ou externe comme les parisiens).

6. Conclusion

Au cours de mon stage, j'ai dû réaliser deux projets informatiques : l'ajout d'un critère de recherche à une pop-up et la gestion de l'indisponibilité d'un service web. Pour cela, il m'a fallu étudier le code source de l'application déjà existante mais également me renseigner sur « le cycle en V » afin de mieux organiser mon projet.

J'ai ensuite dû me documenter sur les librairies nécessaires au fonctionnement du service web de l'application dont la maintenance m'avait été confiée. Lors de mes recherches, mes connaissances de l'Anglais ont été primordiales.

Il m'a ensuite été demandé de rédiger une spécification fonctionnelle, mais aussi d'effectuer différents tests sur les projets que j'ai été amené à réaliser. Ceci m'a permis de découvrir les obligations et contraintes qu'un développeur doit savoir respecter lors de la réalisation d'une tâche.

Pour conclure, ce stage en milieu professionnel m'a permis de découvrir, plus en profondeur, les limites et contraintes du métier de développeur, mais aussi de renforcer mes acquis et apprendre de nouvelles notions.