

LP SIL – Concepteur-Développeur
en Environnement Distribué
2017 - 2018

IUT

Robert Schuman

Institut universitaire de technologie

Université de Strasbourg



PROUST

DOCUMENTATION DU PROJET

COMPLEMENTS

GROUPE « OVERKILL LEGACY »

COUTY	Robin
CROCHEMORE	Thomas
KOEHL	Dylan
PÉQUIGNOT	Ysaline
WEIBEL	Lucas

Table des matières

Table des matières	1
Déploiement	2
Préparatifs.....	2
Machine virtuelle	2
Packages APT.....	2
Installation et configuration Tomcat.....	2
Configuration MySQL et MongoDB.....	4
Récupération du dépôt Git et configuration.....	4
Déploiement de l'archive et lancement.....	4
Ajout des procédures MySQL.....	4
Mise en place du script Python	5
Ajout des tunnels SSH (si besoin).....	5
Réflexions – Moteur de normalisation v2.....	6
Conversion de données : un processus redondant.....	6
Généraliser le calcul : étude de ScriptEngine.....	6
Modèle de données	7
Mise en place de l'algorithme.....	10
Annexe – Table des illustrations	12
Annexe – Index.....	12

Déploiement

Le déploiement de l'application nécessite un certain nombre de ressources avant de pouvoir l'utiliser pleinement.

Préparatifs















Machine virtuelle

Vous devez disposer d'une machine Unix, de préférence chargée avec une image Debian Stretch (version 9). L'image est disponible sur le site officiel Debian, sous la forme d'un .iso. Après avoir créé, si nécessaire, une machine virtuelle et chargé l'image, configurez votre machine habituellement.

NOTE : Si vous souhaitez déployer en local ou sous Windows, installez XAMPP et passez à la section « Récupération du dépôt Git et configuration ».

Packages APT

Avant d'installer les dépendances, pensez à faire un `sudo apt-get update` et un `sudo apt-get upgrade`. La liste des packages APT à installer est la suivante, dans l'ordre :

-  `apt-transport-https`
-  `build-essential`
-  `ca-certificates`
-  `apache2`
-  `autossh`
-  `default-jre`
-  `default-jdk`
-  `git`
-  `mysql-common` (charge une version de MariaDB)
-  `mysql-server` (idem)
-  `mongodb`
-  `python`
-  `python3`
-  `maven`

Installation et configuration Tomcat

Téléchargez l'archive Tomcat en utilisant la commande suivante :

```
wget http://www-us.apache.org/dist/tomcat/tomcat-9/v9.0.8/bin/apache-tomcat-9.0.8.tar.gz
```

Après avoir récupéré le contenu, créez un répertoire `/opt/tomcat9` et déplacez le contenu de l'archive dans ce répertoire. Ensuite, naviguez dans `tomcat9/conf` et modifiez `tomcat-users.xml`. Ajoutez un utilisateur ayant les droits d'accès à la page « manager » de Tomcat :

```
<role rolename="manager"/>
<role rolename="manager-gui"/>
<role rolename="admin"/>
<user username="*****" password="*****" roles="admin,manager,manager-gui"/>
```

Modifiez ensuite le fichier context.xml dans /opt/tomcat9/webapps/manager/META-INF comme suit :

```
<Context antiResourceLocking="false" privileged="true" >
  <!--
    <Valve className="org.apache.catalina.valves.RemoteAddrValve"
      allow="127\.\d+\.\d+\.\d+|:1|0:0:0:0:0:0:0:1" />
  -->
  <Manager
sessionAttributeValueClassNameFilter="java\.lang\.(?:Boolean|Integer|Long|Number|string)
|org\.apache\.catalina\.filters\.CsrfPreventionFilter\$LruCache(?:\$1)?|java\.util\.(?:
:Linked)?HashMap"/>
</Context>
```

Ensuite, créez un nouvel utilisateur nommé tomcat qui se chargera de faire fonctionner le serveur. Ajoutez-le à un nouveau groupe nommé tomcat. Changez son mot de passe avec `sudo passwd tomcat`.

Cet utilisateur doit avoir une entrée dans visudo, mais celle-ci sera limitée à un seul groupe de commandes pour éviter les débordements :

```
tomcat ALL=/usr/sbin/service tomcat9 *
```

Créez un fichier dans /etc/systemd/system nommé tomcat9.service, et ajoutez le contenu suivant :

```
[Unit]
Description=Tomcat 9 instance
After=network.target

[Service]
Type=forking
User=tomcat
Group=tomcat

Environment=CATALINA_PID=/opt/tomcat9/tomcat8.pid
Environment=TOMCAT_JAVA_HOME=/usr/bin/java
Environment=CATALINA_HOME=/opt/tomcat9
Environment=CATALINA_BASE=/opt/tomcat9
Environment=CATALINA_OPTS=
Environment="JAVA_OPTS=-Dfile.encoding=UTF-8 -Dnet.sf.ehcache.skipUpdateCheck=true -
XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled -XX:+UseParNewGC -
XX:MaxPermSize=128m -Xms512m -Xmx512m"

ExecStart=/opt/tomcat9/bin/startup.sh
ExecStop=/bin/kill -15 $MAINPID

[Install]
WantedBy=multi-user.target
```

Sauvegardez le fichier et rechargez systemctl:

```
sudo systemctl daemon-reload
sudo systemctl start tomcat9
sudo systemctl enable tomcat9
```

Donnez les droits globaux sur le répertoire /opt/tomcat9 et ses enfants à l'utilisateur tomcat :

```
sudo chown -R tomcat:tomcat tomcat9/
```

Pour lancer le service, connectez-vous avec le nouvel utilisateur avec la commande `sudo su tomcat -s /bin/bash`, puis exécutez :

```
sudo service tomcat9 start
```

Configuration MySQL et MongoDB

Créez une base de données et ajoutez un utilisateur administrant les bases MongoDB et MySQL. Retenez ses identifiants de connexion, ils vous seront utiles par la suite.

Récupération du dépôt Git et configuration

Vous pouvez récupérer le dépôt Git en clonant ce dernier sur votre machine. Une fois ceci fait, il vous faut configurer l'application avant son déploiement.

Vous trouverez dans `src/main/resources` un certain nombre de fichiers. Les deux fichiers à modifier sont les fichiers `hibernate_*.cfg.xml`, qui contiennent la configuration Hibernate utilisée par le serveur. Les éléments importants sont les suivants :

```
<property name="connection.url">jdbc:mysql://host:port/database</property>
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>

<property name="connection.username">username</property>
<property name="connection.password">password</property>
```

Les éléments à modifier sont `connection.url`, `connection.username` et `connection.password`.

Passez ensuite dans le fichier `proust.properties` situé dans `src/main/resources` et éditez la configuration pour qu'elle corresponde à celle de votre serveur (MongoDB et Hibernate).

Déploiement de l'archive et lancement

Pour déployer l'archive, placez-vous à la racine du projet Git puis entrez la commande suivante :

```
mvn clean install -DskipTests
```

Ceci créera une archive WAR exportable sur Tomcat.

Pour exporter l'archive, deux solutions possibles :

- Si vous déployez depuis un navigateur web, connectez-vous au serveur Tomcat et naviguez sur la partie « Manager App ». Entrez vos identifiants. Enfin, vous trouverez une section pour exporter un fichier. Ajoutez le WAR précédemment créé et validez.
- Si vous déployez depuis la ligne de commande, déplacez l'archive `.war` dans `/opt/tomcat9/webapps`. Tomcat se chargera de décompresser celle-ci. Cependant, pour démarrer l'application, vous devez aussi passer par la « Manager App » sur Tomcat par navigateur.

Ajout des procédures MySQL

Dans les dossiers `/sql/sprint3` et `/sql/sprint3/new`, vous trouverez des scripts SQL à exécuter sur la base MySQL. Le script à exécuter est `/sql/index.sql`.

Pour information, voici l'ordre d'exécution des scripts :

1. `truncate_date.sql`

2. stats_avg_rt.sql
3. stats_proc_backup.sql
4. stats_proc_compute_avr_rt.sql
5. stats_proc_compute.sql
6. stats_proc_scoring.sql
7. stats_values_forecast.sql
8. jump_all_rt.sql
9. stats_values_distance.sql
10. stats_proc_backup_jump_all_rt.sql
11. stats_proc_backup_values_distances.sql
12. stats_proc_compute_jump_all_rt.sql
13. stats_proc_compute_values_distances.sql


Mise en place du script Python

Le script Python se trouve dans le dossier `/scripts` à la racine du projet. Ce script devra être exécuté avec l'aide de l'outil `cron`, fourni sur les machines Unix. Entrez `sudo crontab -e` et entrez ceci :

```
0 0,6,12,18 * * * /usr/bin/python2.7 <répertoire projet>/scripts/proust-dumpscript.py
0 0,6,12,18 * * * curl localhost:8080/<nom de l'archive WAR>/api/weatherdata/compute
```

Sauvegardez les modifications.

Ajout des tunnels SSH (si besoin)

Installez  `autossh` si besoin et configurez des tunnels SSH pour accéder au service Tomcat. Cette partie ne sera pas détaillée car trop dépendante de l'architecture mise en place.

Réflexions – Moteur de normalisation v2

Nous avons réfléchi à un algorithme permettant à l'utilisateur d'ajouter une API dynamiquement. Cet algorithme n'est pas implémenté, mais nous vous fournissons nos réflexions dessus.

Conversion de données : un processus redondant

Pour le moment, notre ajout d'API se fait en rajoutant du code : nous avons un « parser » (convertisseur) par API, convertissant une donnée brute en modèle commun de deux façons : une pour la valeur observée (délai = 0), et une pour la prévision.

Pour chaque processus, nous avons à récupérer une ou plusieurs valeurs. Nous avons remarqué un comportement semblable pour chaque API. Pour une valeur observée (délai = 0), il suffit de récupérer la propriété dans l'objet (par exemple : `main.currentObservation.temp` pour la température).

Pour une prévision, les APIs nous envoient une liste de prévisions : on récupère l'élément de la liste selon la date de prévision (correspondant au délai désiré), puis on récupère la propriété de l'objet selon l'élément que l'on aura filtré. La date peut être fournie sous différentes formes (timestamp en secondes, timestamp en millisecondes, isoDate). (exemple : `main.forecast[n].epoch` pour la date → on filtre l'époque à l'heure désirée, `main.forecast[i].temp` pour la température, où `i` correspond à l'index de l'élément que l'on aura filtré).

Nous avons aussi remarqué que la récupération d'une propriété ne suffisait pas dans plusieurs cas :

- la propriété n'est pas dans l'unité désirée dans le modèle commun, on la convertit (ex : de Kelvin à Celsius)
- la propriété n'est pas donnée directement et nécessite un calcul supplémentaire selon d'autres propriétés (ex : on a la température minimum et maximum, on fait la moyenne des deux valeurs)

On peut donc aussi avoir une combinaison des deux, ou même un calcul qui nécessite le résultat d'un autre calcul.

Si l'on prend en compte le fait qu'une conversion d'unité n'est qu'un calcul à 1 paramètre, ou encore la conversion de date est aussi un calcul, et que le calcul identité retourne le paramètre donné (pour les calculs ne nécessitant aucun paramètre), nous pouvons généraliser ça.

Généraliser le calcul : étude de ScriptEngine

Nous pouvons imaginer que chaque calcul serait une classe qui implémenterait la même interface, mais si nous choisissons cette solution, le même problème viendrait à nous : si une nouvelle formule de calcul doit être implémentée pour une nouvelle API, nous devons rajouter du code. Ce n'est pas le but.

En Java, nous avons donc étudié la librairie Java ScriptEngine. Ceci nous permet d'évaluer une chaîne de caractère avec des paramètres en Javascript ou Ruby (ici, du javascript). Nous pouvons donc, à partir d'une chaîne simple, faire un calcul. Par exemple, le calcul de moyenne : `"(max + min)/2"`. Nous pouvons aussi faire un calcul sur un timestamp : `"new Date(t*1000)"`, qui retournerait une date.

Nous pouvons aussi récupérer une propriété depuis un objet JSON : `"main.current.temp"`, ou encore une liste d'objets `"main.forecasts"`.

Le fait de pouvoir effectuer de tels calculs par le biais de chaînes de caractère nous offre d'énormes possibilités : le stockage d'un calcul en base de données. Avec ceci, nous pouvons nous assurer, avec un modèle de données et un algorithme réfléchis, de la générique de l'ajout d'une nouvelle API.

Modèle de données

Maintenant que nous avons les solutions techniques, nous allons modéliser ces calculs dans un modèle de donnée.

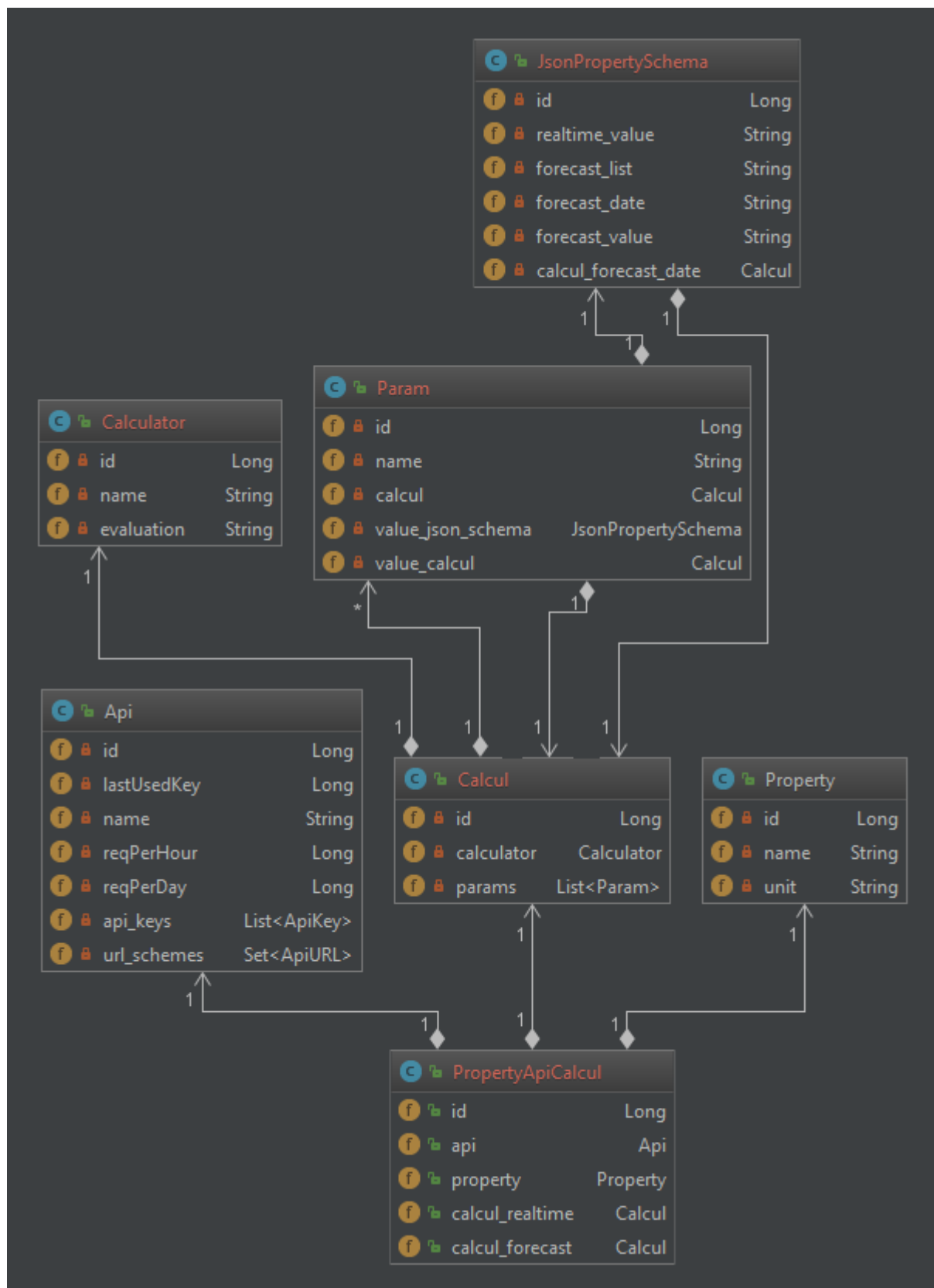


Figure 1 - Modèle de données des calculs

Pour comprendre le modèle de données nous allons comparer ce modèle à de simples fonctions arithmétiques.

Le `calculator` (calculateur) a un nom et une formule à évaluer, c'est l'équivalent d'une fonction mathématique.

Soit le calculateur de nom `"conversion_kelvin_to_celsius"` et d'évaluation `"x-273.15"`, la fonction mathématique équivalente est `conversion_kelvin_to_celsius(x) = x-273.15`. Le calcul a un calculateur et une liste de paramètres, c'est donc équivalent à l'appel d'une fonction.

Nous nous consacrerons au paramètre par la suite. Pour l'exemple, nous prendrons une constante. Pour un calcul ayant un calculateur `conversion_kelvin_to_celsius` et ayant comme paramètre `x = 300.15`, nous avons l'équivalent de `conversion_kelvin_to_celsius(300.15)`. Ici la valeur vaut 27.

Un paramètre peut être soit une constante, soit le résultat d'un autre calcul. Pour une constante, on lui fournit le schéma de la propriété (`JsonSchemaProperty`) qui définit comment on récupère la propriété dans la donnée brute de l'api. Dans tous les cas, il a un nom, le même que le nom de l'évaluation.

Nous avons eu le cas d'une constante, prenons le cas d'un paramètre de calcul. Imaginons que nous avons 2 propriétés `temp_min` et `temp_max`, et que nous en voulons une moyenne. Or, ces propriétés sont données en kelvin. Nous avons donc besoin de 2 calculators, l'un valant de nom `"conversion_kelvin_to_celsius"` et d'évaluation `"x-273.15"`, l'autre de nom `"moyenne"` et d'évaluation `"(max+min)/2"`.

Nous avons alors 2 fonctions mathématiques :

- `conversion_kelvin_to_celsius(x) = x-273.15`
- `moyenne(min,max) = (max + min)/2`

Prenons les propriétés `temp_min = 297.15` et `temp_max = 300.15`. Nous définissons 2 paramètres, l'un de nom `"max"` et de valeur 300.15 et l'autre de nom `"min"` et de valeur 297.15. (Nous verrons par la suite comment récupérer ces valeurs).

Définissons un premier calcul ayant pour calculateur `"moyenne"` et pour paramètre les deux que l'on vient de citer. Nous avons un premier appel qui sera `moyenne(297.15,300.15)`. Définissons un troisième paramètre de nom `"x"` et pour valeur, le résultat du calcul précédent.

Le second calcul aura pour calculateur `conversion_kelvin_to_celsius` et pour paramètre le paramètre `x` précédemment cité. Nous aurons alors, en décomposant, un appel de fonction comme celui-ci :

```
conversion_kelvin_to_celsius(moyenne(297.15,300.15))
```

Le schéma d'une propriété (`JsonSchemaProperty`) sera rentré par l'administrateur, connaissant au préalable comment récupérer la propriété dans l'objet JSON de la réponse. Pour une valeur observée (`delai = 0`), le champ `realtime_value` sera rempli, par exemple `"main.current.temp"` ou indirectement `"main.current.temp_min"` et `"main.current.temp_max"`.

Pour une valeur de prévision, il lui faudra en premier lieu renseigner comment récupérer la liste des prévisions, par exemple `"main.forecasts"`, cela sera renseigné dans `forecast_list`.

Il devra par la suite renseigner comment récupérer la date dans chaque élément de la liste, par exemple `"hourly.epoch"`. Cela sera renseigné dans `forecast_date`. Il devra renseigner le calcul de conversion de date (timestamp en milliseconde, secondes, isodate...), qui sera renseigné dans `calcul_forecast_date`. Ensuite, il devra

renseigner comment récupérer la propriété dans l'élément de la liste (que l'on aura filtré par date), qui sera dans `forecast_value`, par exemple `"temp_min"`.

On lie chaque propriété de l'API à un unique calcul (`PropertyApiCalcul`) pour en définir le résultat final. Ainsi, on pourra combiner les calculs pour en donner un seul résultat. Dans une autre section, l'administrateur pourra définir des calculateurs.

Dans la section d'ajout d'API, on pourra, en plus des éléments d'URL, renseigner l'ajout de propriétés, un choix multiple entre plusieurs calculateurs affichés par leur nom, définir les paramètres en constante ou en utilisant un autre calculateur, jusqu'à arriver récursivement à des uniques constantes définies par leur schéma JSON. Nous partirons du principe que, pour aucun calcul, l'utilisateur ne fournira le calcul identité avec pour paramètre le schéma JSON correspondant.

Mise en place de l'algorithme

Avec ce modèle de données, nous pouvons mettre en place l'algorithme pour récupérer la propriété d'une API. Voici un diagramme permettant de le visualiser :

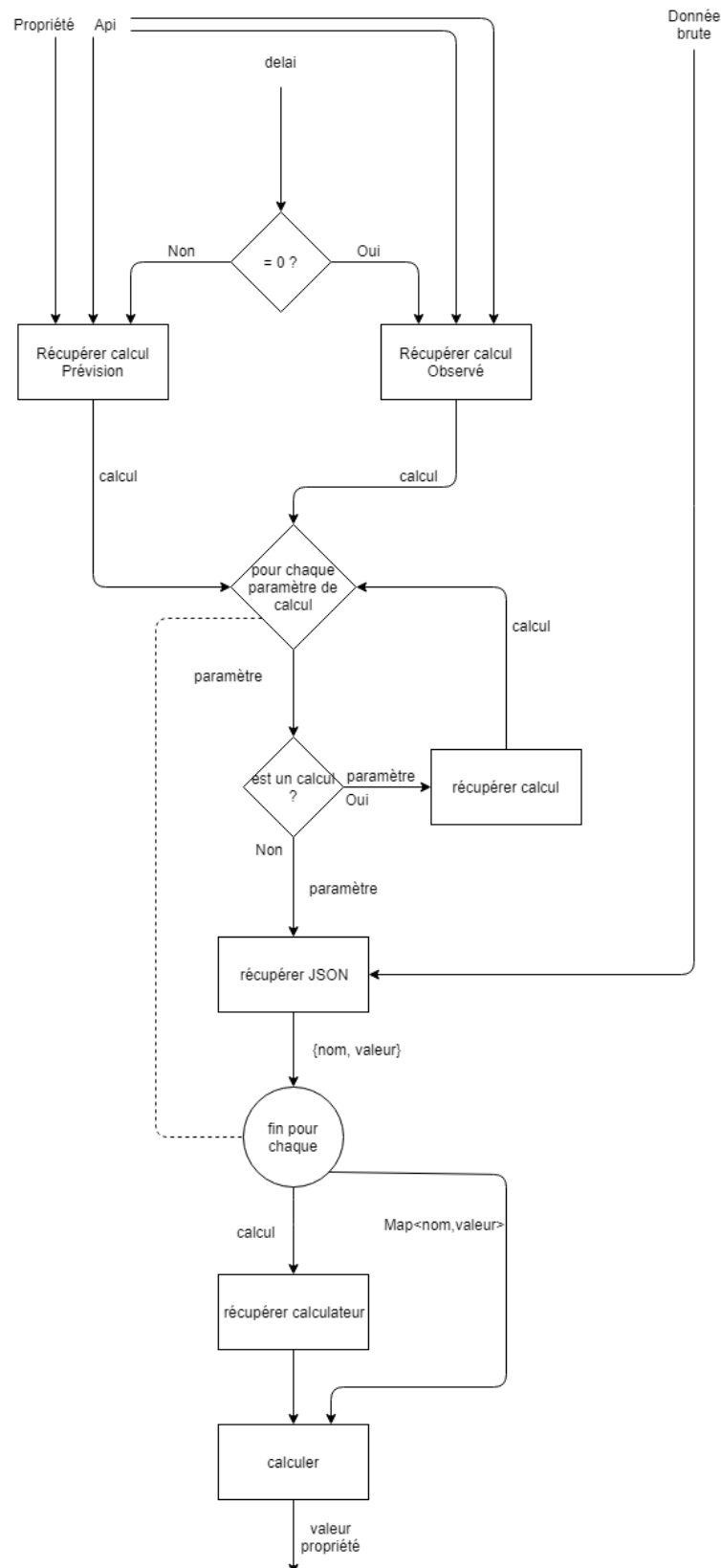


Figure 2 - Processus du nouveau moteur pour récupérer une propriété

Nous avons en entrée de notre algorithme la propriété, l'API, le délai, et la donnée brute de l'API. La première étape consiste à récupérer le calcul principal. Pour cela, on récupère l'entité PropertyApiCalcul qui en possède deux : un pour la valeur observée, un pour la valeur de prévision. On fait donc une condition sur le délai pour savoir quel calcul récupérer.

Pour chaque paramètre du calcul, on récupère sa valeur dans une Map avec pour clé le nom du paramètre. Pour récupérer la valeur d'un paramètre, deux cas s'imposent : si la valeur est une constante, on la récupère depuis son schéma JSON, sinon on récupère récursivement la valeur du calcul correspondant.

Pour récupérer la valeur du schéma JSON, deux cas s'imposent : si la valeur est une valeur observée, on récupère sa propriété grâce à ScriptEngine, par exemple `"main.current.temp"` avec pour paramètre l'objet JSON de la donnée brute de l'api passée en paramètre. Si la valeur est une valeur prévisionnelle, cela se passe en plusieurs étapes :

- On récupère la liste des prévisions grâce à ScriptEngine
- On filtre l'élément de la liste selon la date voulue
- Pour chaque élément, on récupère la date de la donnée brute à l'aide de script engine
- On récupère le calcul correspondant à la conversion de date
- On évalue le calcul
- S'il correspond à date+délai à l'heure prêt, on retourne cet élément
- Sinon, on continue à parcourir la liste
- À partir de l'élément récupéré par le filtre, on récupère la propriété `forecast_value` à l'aide de scriptEngine

Une fois les paramètres récupérés récursivement, on procède au calcul : on récupère le calculator du calcul, puis on lui applique les paramètres de la Map construite précédemment. Cette étape est faite récursivement pour chaque paramètre devant faire un calcul.

La dernière étape du calcul nous donne la valeur de la propriété voulue convertie dans le modèle commun.

Annexe – Table des illustrations

Figure 1 – Informations essentielles du rapport SonarQube..... Erreur ! Signet non défini.

Figure 2 - Rapport de couverture de code JaCoCo Erreur ! Signet non défini.

Annexe – Index

ChatOps _____	5	StatusCake _____	5, 10
couverture _____	8, 11	test _____	1, 3, 5, 7, 8, 10
ergonomie _____	4	tests _____	1, 2, 4, 7, 8, 9, 10
MongoDB _____	5, 7	Tomcat _____	5, 10
MySQL _____	5, 7, 8		