**TP Webmapping Client-Serveur**

Auteur : Emmanuel Fritsch

Version 1 : décembre 2018

Version 2 : novembre 2022

# Méthode

Nous allons créer un client et un serveur pour publier les données de tremblements de terre. Une possibilité serait de créer un flux WFS, et de le lire avec un client WFS. Nous allons prendre un autre chemin : la définition d’un format de web service propre à notre application.

Nous allons découper le travail en trois blocs : d'abord la constitution des données dans PostGIS (c'est ce que nous avons fait dans le cours PostGIS), ensuite la programmation du serveur, enfin la programmation du client.

Pour que ces trois tâches puissent être exécutées indépendamment les unes des autres, la programmation du serveur et celle du client sont précédées chacune d'un petit travail de constitution d'un jeu test : ainsi, on peut tester le client sans disposer du serveur, et tester le serveur sans disposer des données.

Naturellement, nous allons écrire nos tests unitaires.

## Affichage cartographique

Vous allez faire vos affichages cartographiques avec MapQuest. MapQuest est un client cartographique, proche dans le principe de l'API Google Maps. Vous vous appuierez sur la documentation en ligne. Pour commencer, allez prendre une clef sur le service MapQuest.  
(Attention, pour un usage ouvert, gratuit, l'utilisation d'une API cartographique n'offre pas plus de garantie que celle d'un web service. Ainsi, au cour de l’été 2018, et presque sans préavis, l'API Google Maps a fortement durci ces conditions de gratuité - obligeant nombre d'utilisateurs à trouver en urgence une alternative, ou à perdre le service).

Si vous n’arrivez pas à utiliser MapQuest, vous utiliserez directement LeafLet, avec une couche WMS MondeGadm issue des TP précédents.

## Structuration du code

A la racine de notre projet, nous allons mettre quatre répertoires : "client" et "serveur", "tu\_client" et "tu\_server". Les deux premiers manifestent la forte séparation logicielle attendue entre le client et le serveur. Dans un contexte professionnel, cette séparation permet par exemple de faire travailler les équipes de manière plus modulaire, l’une sur le client, l’autre sur le serveur, l’interface entre les deux étant réalisée par la définition du format des web services.

Dans notre première architecture, nous allons récupérer toutes les données lors du lancement du client. Dans une seconde version, plus adaptée à nos données, nous rechargerons les données à chaque modification de la vue.

# Première architecture

Le fonctionnement proposé est le suivant :

* Lorsque le client se charge, il charge en même temps toutes les données et les affiche toutes
* Le web service nécessaire pour le fonctionnement de ce client aspire donc toutes les données

Comme nos tremblements de terre sont trop nombreux, nous nous limiterons ici aux 100 objets de plus forte magnitude (par exemple en reprenant la vue tdt\_magni\_sup).

## Définition du format de web service

Pour chaque tremblement de terre, nous voulons récupérer toutes les données que nous avons manipulées dans notre TP PostGIS :

* Les colonnes initiales
* Le plus proche voisin, la nature terrestre du tremblement, sa nature côtière, la distance à la cote.

Notez qu’on ne chargera pas les géométries des tremblements de terre (la colonne geom), qui n’est pas en format texte, et que les latitudes et longitudes feront très bien l’affaire.

Le format de la réponse est un fichier json, avec un tableau de tremblements de terre, chaque objet tremblement de terre étant constitué des différents attributs : magni, prof, lat, lng, ppv, terr, cot, dist.

## Programmation du client

* Ecrivez un fichier simulant les données : ouvrez un jsoneditor, et rédigez avec cette interface un fichier de trois faux tremblements de terre, avec les attributs proposés.
* Enregistrez le code json dans un fichier *my\_tdt1.json*, et posez ce fichier dans le répertoire server. En faire un second, avec des localisations de tdt différentes, nommés *my\_tdt2.json*. Ces fichiers vont nous servir de base de test.
* Ecrivez deux boutons de tests unitaires : lorsque je clique sur le premier bouton, les tremblements de terre du premier fichier s'affichent à leur bonne place. Un clic sur le deuxième bouton détruit les tdt existants, et affiche ceux du deuxième fichier.
* Ecrivez un programme javascript qui envoie une requête ajax vers le serveur, pour charger my\_tdt1.json.
* Au retour de la requête, on affiche les tdt nouvellement arrivés, et on détruit les anciens.
* Lancez les tests unitaires. Conclure.
* Programmer la fonctionnalité suivante : Un bouton permet, lorsqu'il est cliqué, de recharger les tdt sur l'emprise courante de la carte.
* Ajouter le chargement automatique de *my\_tdt1.json* lors de l’ouverture du client.
* Programmez la fonctionnalité suivante : Lorsqu'on clique sur un tremblement de terre, toutes les informations sur ce tremblement de terre s'affichent dans une div extérieure à la carte (par exemple à droite de la carte).

## Programmation du serveur

* Installation du serveur : wampserver 64 bit version 3 – sur le disque D.
* Constitution du jeu test sous PostGIS. Ce travail est directement préalable à la programmation serveur. Pour aller plus vite, j'ai fait le travail à votre place. Il vous reste juste à le relire et à le débuguer. C'est le fichier test\_tdt.sql.
* Ecrivez le test unitaire. Vérifiez auprès de votre enseignant ou de vos pairs.
* Ecrivez un programme PHP *all\_100\_big\_tdt.ws.php* qui extrait les 100 tremblements de terre de plus forte magnitude …
* … et qui les renvoie au format json.
* Débuguez en vous aidant des tests unitaires.

## Interfaçage

* Pour le client, il suffit de remplacer dans AJAX l'adresse *my\_tdt1.json* par *all\_100\_big\_tdt.ws.php*.
* Et pour le serveur ?
* Testez.

# Utilisation d’un WS avec paramètres - extraire des données

L’architecture adoptée dans la partie précédente consiste à télécharger toutes les données nécessaires à l’application lors de l’ouverture du client. Mais télécharger toutes les données n’est pas toujours une bonne idée :

* D’une part si les données sont trop nombreuses, vous allez trop fortement solliciter les infrastructures de communication du serveur, mais aussi les capacités de traitement du client.
* D’autre part ces données peuvent être patrimoniales : leur détenteur peut avoir des réticences à les voir sortir en masse.

De fait, dans notre premier essai ci-dessus, nous n’avons pas téléchargé les vingt-cinq milliers de tremblements de terre.

Nous allons reprendre l’architecture du système, pour lui conférer un fonctionnement moins dangereux :

* Le client envoie une requête sur l’emprise de visualisation, en spécifiant un nombre limité de tremblements de terre. Si le nombre d’objets sur l’emprise est trop important, ceux de plus forte magnitude seront privilégiés.
* On peut choisir la plage des dates des tremblements de terre. Un bouton apparaît alors, qui propose de recharger les données.
* A chaque fois qu’on change d’emprise, on relance la requête,
* Au retour de la requête, on ajoute les objets reçus en réponse, puis on efface les anciens.
* On ne récupère pas toutes les colonnes sémantiques, mais seulement celles qu’on veut afficher.

Pour commencer, faites une copie d’archive du travail précédent.

## Définition du web service

Pour spécifier les tremblements de terre à choisir, il faut passer des paramètres au web service. Ces paramètres sont :

* **lng\_min, lat\_min, lng\_max, lat\_max** : les coordonnées WGS84 de la boîte englobante de la requête. Par défaut : -180, -90, 180, 90.
* **date\_min, date\_max** : les dates définissant l’intervalle de temps d’occurrence des événements requis. Par défaut : aucune limitation sur les date.
* **magni\_min, magni \_max** : les bornes inférieures et supérieures des magnitudes appelées. Par défaut : aucune limitation sur les magnitudes.
* **columns** : une chaîne de caractère listant les noms des attributs à servir, parmi : magni, prof, ppv, terr, cot, dist. Les noms sont séparés par des virgules. Les noms non conformes sont ignorés. La valeur par défaut est : "magni,prof". Les colonnes lat et lng sont toujours servies.
* **limit** : le nombre maximum de tremblements de terre servis. Par défaut : 100. Les bornes de ce paramètre limit sont 1 et 300, i.e. si le paramètre limit est supérieur à 300, la valeur prise en compte est ramenée à 300, et de même, une valeur inférieure à 1 est ramenée à 1.

Ces paramètres seront passés en mode GET. Le service s’appellera *my\_tdt.ws.php*.

Le format de retour sera le même que dans la partie précédente.

## Les tests unitaires et la documentation du service

Puisque le service est en mode GET, les paramètres sont passés dans l’adresse de la requête, en entête de la requête HTTP (si la différence entre mode GET et POST n’est pas parfaitement claire pour vous, je vous demande de combler cette lacune : voir les liens proposés à la fin de ce document).

* Pour tester le service, rédigez des adresses de service, avec les paramètres séparés par des '&', selon le modèle :   
   <http://localhost/my_tdt.ws.php?param1=val1&param2=val2&param3=etc>.

avec des valeurs conformes pour les paramètres.

* Chaque test unitaire doit tester une catégorie de paramètres.
* Lorsqu’on active une adresse, le client reçoit une réponse en json. Il faut afficher cette réponse :
  + sous la forme du fichier text/json dans une simple div
  + sous la forme des localisations des tremblements de terre une API cartographique simple (sans widget de navigation).

La page des tests unitaires peut en constituer en même temps la documentation, avec la description des différents formats d’entrée et de sortie. Adopter cette politique de documentation augmente la facilité de prise en main du produit, et en facile ainsi la diffusion.

## La programmation du serveur

Copiez le fichier *all\_100\_big\_tdt.ws.php* en *my\_tdt.ws.php*.

Adaptez-le pour assurer la prise en compte des paramètres.

## Les tests unitaires du client

Les tests unitaires du client, doivent permettre de vérifier le bon fonctionnement du client alors qu’on ne dispose pas encore du web service avec lequel il devra s’interfacer.

Pour tester le client, il faut :

* Tester la réaction du client aux retours du serveur, en utilisant des retours statiques. Nous l’avons fait dans la première partie.
* Tester la réaction du client aux manipulations d’interface qui ne provoquent pas d’appel au serveur. Si ces manipulations ne provoquent pas de requêtes vers le serveur, mais seulement des modifications de l’interface, il faut alors les documenter. Sinon, voir point suivant.
* Tester la bonne expédition des requêtes, avec leurs paramètres vers le serveur. Pour cela, il faut, dans ce lot de tests unitaires,
  + soit bloquer l’émission de la requête ajax, et à la place afficher la requête avec la liste des paramètres,
  + soit implémenter un pseudo-serveur qui se borne à renvoyer la liste des paramètres mis en entrée
  + soit, à défaut, indiquer à l’utilisateur de vérifier la bonne émission de la requête depuis le navigateur, dans la console réseau.

Dans tous les cas, les tests unitaires doivent être documentés.

## Implémentation du client

Au client programmé dans la première partie, on va ajouter :

* un double slider qui permet de saisir les bornes de l’intervalle des dates des tremblements de terre qu’on affiche.
* un double slider qui permet de faire la même chose avec les magnitudes
* Un slider qui permet de changer le nombre maximal d’objets
* Des checkbox pour choisir la variable que l’on représente cartographiquement (profondeur, magnitude, date)
* Des radiobox pour attraper les variables que l’on affiche (profondeur, magnitude, date)
* Un bouton qui envoie la requête lorsqu’on clique dessus

Lorsque tout fonctionne : on souhaite que la requête soit postée dès que la carte subit une transformation (attention, cette fonctionnalité est potentiellement gourmande en bande passante – et en CO2).

Lorsque cela fonctionne, on peut mettre en place un switch entre les deux comportements (voir par exemple la gestion du requêtage cartographique sur l’interface carto de seloger.com)

## Interfaçage

* Dans le client, entrez dans l’appel AJAX l’adresse de votre nouveau web service (my\_tdt.ws.php).
* Et pour le serveur ?
* Testez.
* Documentez !

# Références

Un éditeur json, l’un des meilleurs :

<https://jsoneditoronline.org/>

Mapquest :

<https://developer.mapquest.com/>

Sur les méthodes GET et POST :

<https://www.w3schools.com/tags/ref_httpmethods.asp>

Sur le protocole HTTP :

<https://developer.mozilla.org/fr/docs/Web/HTTP>

Le site seloger.com, une interface de recherche cartographique évoluée :

[https://www.seloger.com/map.htm?projects=2,5&types=2,1&natures=1,2,4&places=[{%22inseeCodes%22:[750111]}]&enterprise=0&qsVersion=1.0&m=search\_results\_img-redirection-search\_results\_map](https://www.seloger.com/map.htm?projects=2,5&types=2,1&natures=1,2,4&places=%5b%7b%22inseeCodes%22:%5b750111%5d%7d%5d&enterprise=0&qsVersion=1.0&m=search_results_img-redirection-search_results_map)

(liens contrôlés le 22 novembre 2022)