SYSTEME D'INFORMATIONS GEOGRAPHIQUES NOMADES

Projet : Structures Judiciaires

GALLET Benoît, HERRMANN Emmanuel, RÉTY Martin

I. Fonctionnement global

L'application fonctionne de la sorte : à son lancement, si le téléphone possède une connexion Internet valide, un téléchargement des json disponibles sur le serveur a lieu pour mettre à jour ceux présents sur le téléphone. L'utilisateur accède ensuite à la carte OpenStreetMap avec dessus les différents points correspondants aux différentes structures judiciaires présentes dans la région. La carte est centrée par défaut sur Paris si c'est la première fois que l'application est lancée, et un service GPS tourne en arrière plan dans l'application en renseignant régulièrement la position de l'utilisateur. Un bouton permettant de recentrer la carte sur la position actuelle de l'utilisateur est disponible, un autre pour rafraîchir la version des json présente sur le téléphone, un bouton pour afficher ou non certains layers, et enfin un dernier permettant d'ajouter un nouvel avocat.

Si un avocat est rajouté, le serveur récupère les données et met à jour la base de données PostGis. Une fois que l'administrateur approuve les changements faits, il lance une commande permettant au serveur Apache Tomcat de récupérer automatiquement les nouveau json générés.

I. Architecture du code

Le code est séparé en deux grandes parties. Notre application s'exécute en grande partie dans une WebView, où la carte OpenStreetMap et les layers sont affichés via OpenLayers. La partie écrite en native correspond aux différentes activités, tâches asynchrones ou autres permettant au javascript de pouvoir s'exécuter et de communiquer avec le serveur.

1. Code natif

Cette partie du code est séparée en deux packages. Le premier est model, qui regroupe trois classes permettant de faire principalement des tâches asynchrones, donc sans mettre à jour la vue.

DownloadFile permet de télécharger les fichiers json au lancement de l'application ou lorsque l'utilisateur veut synchroniser ses données locales avec celles du serveur pour recevoir les dernières mises à jour.

GPSService est un service qui tourne en continu en arrière-plan de l'application et qui récupère les coordonnées géographiques de l'utilisateur (normalement toutes les secondes). Celles-ci sont sauvegardées dans les SharedPreferences en mode private afin qu'aucune autre application ne puisse y accéder. L'activité principale récupère ces données quand elle en a besoin, et si elles existent, cela permet à l'application de se centrer sur la dernière localisation connue de l'utilisateur au lancement.

Enfin, JavascriptConnection définit une suite de méthodes pouvant être appelées à partir du javascript. Comme cela, si à certains moments la Webview a besoin d'avoir des informations, comme lancer une activité telle l'ajout d'un avocat ou avoir les dernières coordonnées géographiques, elle peut récupérer les données voulues.

Le deuxième package est appelé View, car c'est celui qui communique directement avec les différentes vues.

MainActivity est l'activité principale, c'est elle qui lance la WebView, qui appelle les différents services, qui sert de relais entre le javascript et JavaScriptConnection, voire entre les différentes activités/classes/…

AddLawyerActivity est une activité permettant d'ajouter une personne dans la base (comme un avocat, un huissier, etc), et enfin, WebViewActivity est tout simplement l'activité qui gère la WebView.

2. JAVASCRIPT – OPENLAYERS

L'application fonctionne de façon majoritaire dans une WebView avec une couche en JavaScript et OpenLayers. Ainsi, nous avons MobileWebPage.js qui s'occupe d'afficher le fond de carte grâce à OpenStreetMap, et les différents layers que nous avons à partir de fichiers GeoJSON récupérés sur un serveur dans le cas où nous avons une connexion internet, ou directement depuis le téléphone sinon.

Dans ce fichier nous retrouvons des fonctions pour mettre à jour la longitude et la latitude de l'utilisateur, re-centrer la carte sur soi, afficher ou cacher les layers, et enfin la fonction qui sert à créer la carte et les différents layers.

Le deuxième fichier de JavaScript est popup.js, qui permet de gérer les différentes interactions avec l'utilisateur lorsqu'il clique sur la carte et d'afficher une popup lorsque c'est nécessaire. Ainsi lorsque l'utilisateur :

- clique sur un point, une popup apparaît et s'affichent alors les informations du point, tel que son nom, son adresse, son numéro de téléphone, etc.
- clique sur un bouton dans la popup, l'action correspondante est déclenchée comme afficher une page internet, lancer l'application de téléphone avec le numéro, calculer la distance à vol d'oiseau ou bien un itinéraire routier entre l'utilisateur et le point sélectionné avec un affichage correspondant.
- clique sur la carte, deux options sont possibles :
  - si l'utilisateur a au préalable cliqué sur le bouton '+' dans la barre d'outils, alors une popup s'affiche indiquant que l'utilisateur a la possibilité d'ajouter une personne à la base de données à l'endroit cliqué.
  - sinon, on fait disparaître les éventuelles popup ou distance affichées à l'écran.

II. SERVEUR

Le serveur est séparé en deux parties, une mettant à disposition des fichiers json pour que l'application puisse les récupérer, et l'autre reçoit des données des personnes à ajouter à la base. Un script bash permet de lancer le serveur, et à chaque lancement ou demande de l'administrateur, extrait les informations de la base PostGis en shapefiles, convertit ces shapefiles en json, puis les places au bon endroit sur le serveur Tomcat afin qu'ils soient accessibles.

1. Envoi

Le script que nous avons codé, comme évoqué plus haut, se charge de faire automatiquement des tâches répétitives et demandant une action humaine. Il n'y a donc rien à faire de plus, à part éventuellement vérifier que l'export s'est bien passé.

2. Réception

Un script python écoute ce qui arrive sur le port 2512. Si une requête arrive, il la parse, récupère les données correspondant à une personne (nom, prénom, localisation, profession, …) et l'insère dans la base en transformant la localisation en un geom. Une fois que l'administrateur a vérifié les informations ajoutées, il peut lancer l'export en tapant une commande dans le script, qui exporte la base les données et génère les nouveaux json.

III. Problèmes rencontrés

1. La structuration du projet

Notre premier problème a été la structuration du code et l'accès aux données. En effet, nous avons pensé à accéder aux données à partir de Geoserver et d'utiliser les URLs des fichiers Openlayers (WMS) directement dans nos fichiers Javascript. Cette méthode permettait facilement d'afficher, en mode connecté, les données. Cependant, en mode déconnecté, il était nécessaire de récupérer un fichier contenant les informations puis de faire appel à celui-ci. Pour effectuer l'enregistrement des données nous avons choisi l'utilisation du format GeoJSON, qui est également disponible sur Geoserver.
Après avoir réussi à afficher les données via les fichiers GeoJSON nous avons décidé de changer notre structure et de partir sur l'utilisation de ce format pour faire l'affichage de nos layers dans les deux modes d'utilisation. En effet, il était plus simple de garder le même mode d'utilisation d'Openlayers (en mode vectorisé) et de seulement télécharger les nouvelles données quand nous étions en mode connecté.

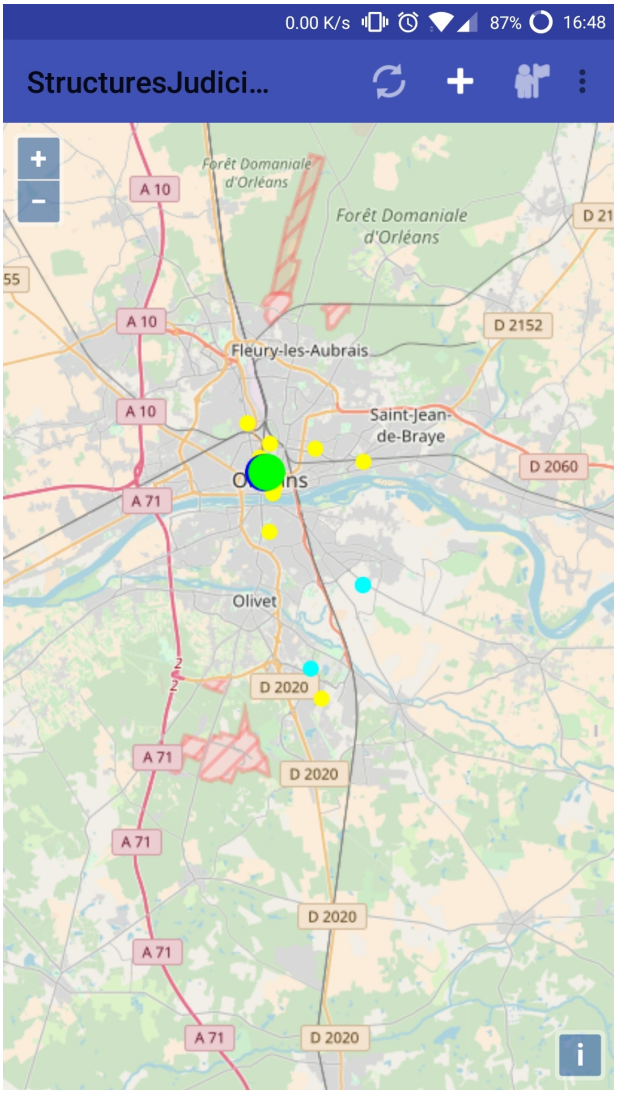
2. Problème sur l'envoi de données entre Java et le serveur

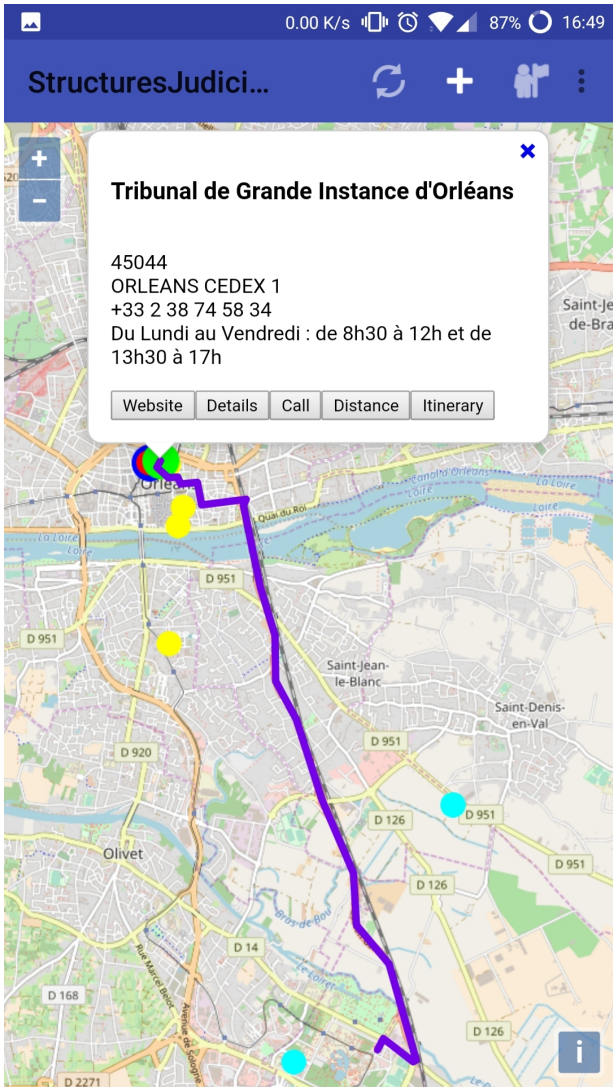
La possibilité de rajouter une personne à la base implique une communication Java → Serveur. Pour ce faire, nous avions opté au début pour le fait d'utiliser un objet HttpUrlConnection, qui nous donnerait ensuite un DataOutputStream sur lequel on pourrait ensuite écrire les informations relatives à la personne à ajouter à la base de données. Cependant, après avoir réussi à établir une connexion via ce procédé, il nous était impossible de réellement communiquer avec le serveur. Nous avons donc dû chercher d'autres options comme utiliser la méthode fetch en JavaScript, utiliser une API pour envoyer un fichier complet, etc. Au final, la solution qui semblait la plus simple était d'utiliser un objet de type XMLHttpRequest d'Ajax, qui nous permet très facilement d'établir une connexion avec un serveur, et de lui envoyer du texte sur un port précis. Au serveur ensuite d'écouter sur ce port pour récupérer les données et les traiter.

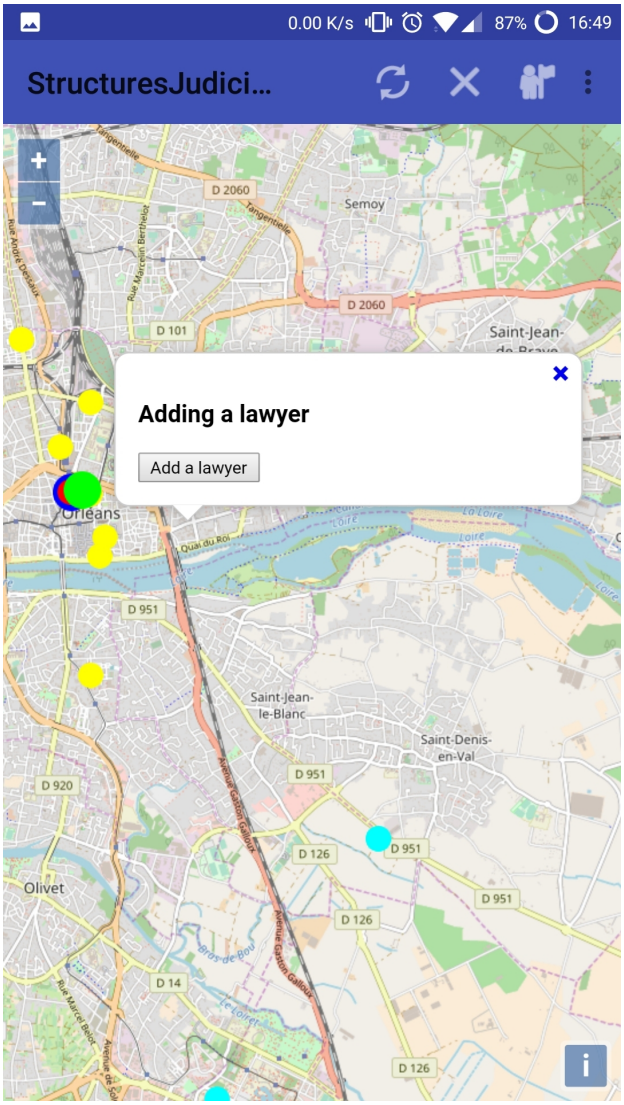
IV. Répartition du travail

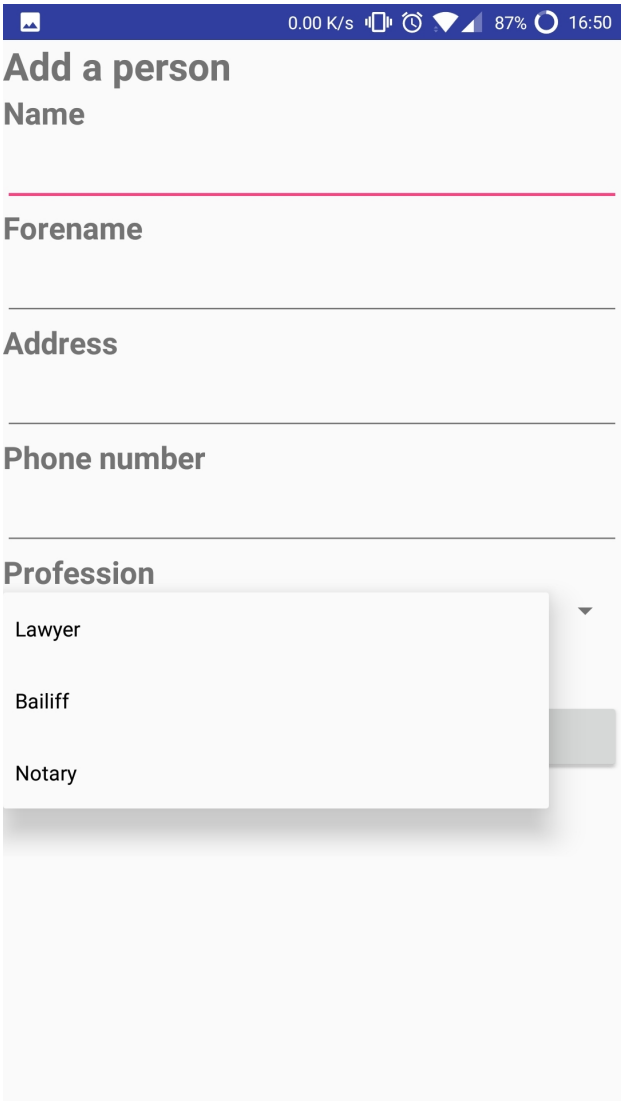
Au début du projet, nous avons travaillé ensemble car nous avions des problèmes pour saisir ce que le projet devait être structurellement parlant. Une fois cela mis au clair pour nous, nous avons travaillé en parallèle, chacun s'occupant d'une tâche ou d'une fonctionnalité, Benoît s'est par exemple occupé de la communication entre le javascript et l'Android, Emmanuel des services GPS et de téléchargement, et Martin de layers et de fonctionnalités OpenLayers. Cela a été rendu possible grâce à l'utilisation d'un dépôt Git, chacun ayant une branche, qu'il mergeait avec master quand la fonctionnalité était opérationnelle. Enfin, ce rapport a été écrit de façon collégiale.

V. Impressions écrans






0.00 K/s 87% 16:49
StructuresJudici...
Adding a lawyer
Add a lawyer
Orléans
Semoy
Saint-Jean-le-Blanc
Saint-Denis-en-Val
Olivet
D 2060
D 101
D 951
D 920
D 126
D 14


0.00 K/s 87% 16:50
Add a person
Name
Forename
Address
Phone number
Profession
Lawyer
Bailiff
Notary