

Le Pigeon Nelson
Projet Prep'Isima 2020 - 2021

Albert Lisa
Veysseyre Julien

7 septembre 2023

Table des matières

1	Introduction	3
1.1	Utilisation du <i>Pigeon Nelson</i>	3
1.2	Fonctionnement du <i>Pigeon Nelson</i>	4
2	Les serveurs statiques	4
2.1	Le Json	4
2.1.1	La synthèse vocale	5
2.1.2	La lecture des fichiers sons	6
2.2	Imprécision de la géolocalisation	6
2.2.1	Le fonctionnement du GPS	6
2.2.2	Résolution du problème de l'effet tunnel	6
3	Les serveurs dynamiques	7
3.1	<i>OpenStreetMap</i>	7
3.1.1	Description	7
3.1.2	Contribution	8
3.2	<i>OverPass Turbo</i>	8
3.2.1	Les différents paramètres	9
3.3	<i>OpenStreetMap</i> et <i>OverPass</i> au service du <i>Pigeon Nelson</i>	10
4	Notre contribution au <i>Pigeon Nelson</i>	11
4.1	Les étapes	11
4.1.1	Première étape	11
4.1.2	Deuxième étape	11
4.1.3	Troisième étape	11
4.2	Le fonctionnement du serveur	12
5	Conclusion	14

1 Introduction

Actuellement en deuxième année de préparation pour intégrer l'école d'informatique ISIMA, nous avons eu l'opportunité de prendre part à un projet. Celui-ci est encadré par Mr. Favreau, un enseignant chercheur en informatique au LIMOS (laboratoire de l'université Clermont-Auvergne). Nous avons eu un peu moins de quatre mois pour comprendre et contribuer à un logiciel en phase de développement se nommant *Pigeon Nelson*. Il est composé à la fois d'une application mobile et d'un ensemble de serveurs. L'idée du *Pigeon Nelson* est assez récente puisque ce n'est qu'en août 2020 qu'elle a émergé. Si vous voulez comprendre l'origine de ce nom, alors imaginez-vous un instant un pigeon chantant "*on the rrrrrroad again*" de Willie Nelson.

1.1 Utilisation du *Pigeon Nelson*

L'idée principale est de proposer du son joué selon l'endroit où l'on est. Son utilisation est simple. Tout d'abord, dans l'application, il faut ajouter un serveur déjà existant dans la base de données des serveurs dédiés au *Pigeon Nelson*. Ensuite, toujours au niveau de l'application, si vous cliquez sur le nom de ce serveur, alors les données de votre position GPS seront transmises si nécessaires et des requêtes seront effectués au niveau du serveur choisi. Enfin, la réponse sera renvoyée au *Pigeon*, qui vous lira le son soit par synthèse vocale, soit en vous faisant écouter un fichier audio préexistant. Pour ajouter un serveur sur l'application, plusieurs manières existent. Il est possible de taper entièrement l'URL dans l'espace de saisie manuelle, ce qui est la méthode initiale. Mais au fur et à mesure des améliorations, des options sont apparues comme le fait de pouvoir cliquer sur un lien ou encore l'utilisation d'un QR Code. Cliquer sur le lien permettra d'ouvrir directement le *Pigeon* et d'y ajouter automatiquement le serveur. Sur l'application, il y a aussi des serveurs pré-intégrés appelés "serveurs publics" qui sont à disposition de tout utilisateur. (voir figure 1)

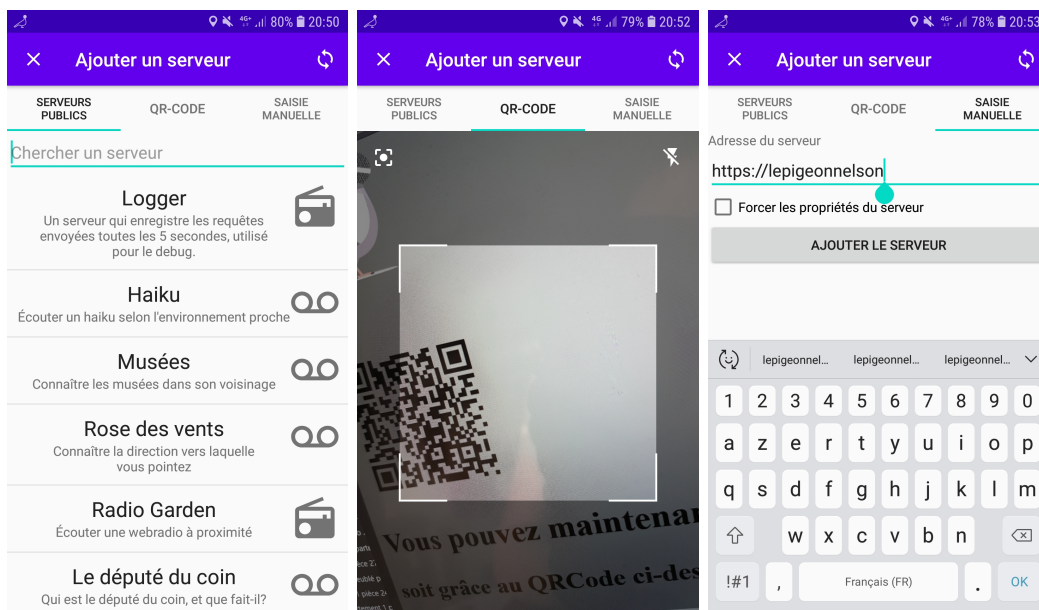


FIGURE 1 – Les différents moyens d'ajouter un serveur au *Pigeon Nelson*

1.2 Fonctionnement du *Pigeon Nelson*

L'application fonctionne en collectant à intervalles réguliers la position, l'orientation et le mouvement de l'utilisateur. Elle consulte ensuite un serveur et celui-ci va par la suite renvoyer un ensemble de messages diffusables conservés dans l'application. Ces messages contiennent certaines conditions (localisation, orientation, ...) dans quel cas si elles sont respectées alors les messages pourront être joués. (voir figure 2)

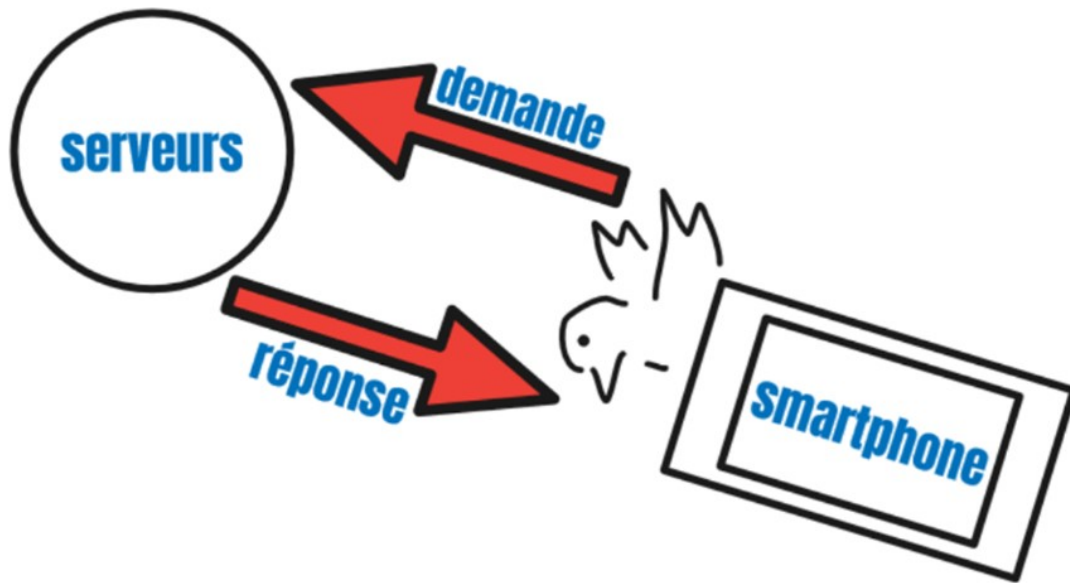


FIGURE 2 – Schéma représentant le fonctionnement du *Pigeon Nelson* de manière simplifiée

2 Les serveurs statiques

Un serveur statique n'exécute pas de codes côté serveur. Il ne va donc ni récupérer des données à stocker et ni les réutiliser.

2.1 Le Json

Le JSON (*JavaScript Object Notation*) est en général utilisé comme langage de transport de données entre le côté client et le côté serveur. C'est un langage de description objet dont le format se présente sous forme de données textuelles comportant soit une liste ordonnée de valeurs, soit un ensemble de couples clés/valeurs. Son principal avantage est qu'il est suffisamment générique et abstrait pour pouvoir représenter n'importe quelle donnée. Cependant, il peut également être utilisé comme moyen de produire du son, et c'est pour cette raison que nous l'avons utilisé dans cette partie.

2.1.1 La synthèse vocale

Au tout début du projet, nous avons commencé à nous familiariser avec le JSON et nous avons créé un serveur permettant d'indiquer à l'utilisateur s'il se situait près d'un endroit précis, ou non. En l'occurrence, il s'agissait du cinéma de notre petit village (confinement oblige, nous étions restreints en termes de distance). Dans ce cas-là, l'usage du JSON est très simple. Il a suffi de récupérer la longitude et la latitude du point correspondant au cinéma, puis nous avons choisi un rayon en mètres autour duquel nous considérons être suffisamment proche du lieu. Ainsi, selon les coordonnées GPS de la position de l'utilisateur, récoltées par le *Pigeon Nelson*, le serveur peut jouer un message par synthèse vocal disant si nous nous situons à proximité du cinéma de notre village.

(voir figure 3) Le message lu sera le champ dont la clé correspond à "txt" et comme indiqué en face de "lang", la synthèse vocale utilisée sera celle lisant le français. Dans le premier cas, le message indiquera que nous sommes proches du cinéma à la condition que nous soyons dans un rayon strictement inférieur à cent mètres par rapport aux coordonnées du cinéma. A l'inverse, si nous sommes à plus de cent mètres du cinéma, le message nous dira donc que nous ne sommes pas à proximité du lieu.

```
[
  {
    "txt": "Vous êtes proche du cinéma de Vayrac!",
    "lang": "fr",
    "priority": 1,
    "requiredConditions": [
      {
        "reference": "distanceTo(44.95498, 1.70241)",
        "comparison": "<",
        "parameter": "100"
      }
    ],
    "forgettingConditions": []
  },
  {
    "txt": "Vous n'êtes pas à proximité cinéma de Vayrac !",
    "lang": "fr",
    "priority": 1,
    "requiredConditions": [
      {
        "reference": "distanceTo(44.95498, 1.70241)",
        "comparison": ">",
        "parameter": "100"
      }
    ],
    "forgettingConditions": []
  }
]
```

FIGURE 3 – Exemple de fichier *.json* permettant de connaître la proximité de l'utilisateur par rapport à un lieu

2.1.2 La lecture des fichiers sons

Nous pouvons également utiliser le format JSON pour lire des fichiers audios préexistant. C'est ce que nous avons pu découvrir en collaborant avec des étudiants géographes travaillant sur le projet *sonographies* qui a été initié en 2015 par Radio Campus Clermont-Ferrand (pour plus de renseignements vous pouvez consulter ce lien : Radio Campus Clermont-Ferrand). Ils nous ont parlé de leur balade sonore "*anecdo – train*" autour de la découverte du quartier de la gare de Clermont, et nous leur avons proposé la création d'un serveur autonome pouvant diffuser différents audios à l'endroit voulu. Cette fois, nous n'avons donc pas utilisé la synthèse vocale mais nous avons intégré des URL correspondant à des fichiers *.mp3* dans le JSON. Ainsi, à partir des coordonnées GPS fournies par les géographes, nous avons pu programmer la diffusion des différents fichiers sons dans des endroits précis de la ville. Cependant, par endroit, plusieurs pastilles sons devaient être joués, il a donc fallu définir laquelle était prioritaire par rapport à l'autre.

2.2 Imprécision de la géolocalisation

Un des principaux problèmes que nous avons rencontré lors de la mise en œuvre de ce serveur est celui relatif aux erreurs d'imprécisions du calcul de la géolocalisation.

2.2.1 Le fonctionnement du GPS

Pour comprendre ce problème, nous allons vous expliquer comment fonctionne le système de positionnement par satellites. A l'heure actuelle, vingt-quatre satellites gravitent autour de la Terre. Au sol, il y a cinq stations qui se chargent d'envoyer en permanence des informations aux satellites afin que ces derniers suivent la trajectoire parfaite. Pour réussir à récolter des données GPS, il faut quatre satellites et un récepteur au sol. Tout d'abord, trois satellites vont servir à définir une position dans l'espace selon les coordonnées (x,y,z). Ensuite, un quatrième satellite va être utilisé afin d'avoir une mesure du temps extrêmement précise à partir de son horloge atomique. Au sol, un récepteur GPS, comme celui placé dans un smartphone, va décoder les signaux reçus seulement s'il capte au moins quatre satellites, et va calculer sa distance par rapport à trois satellites afin d'obtenir sa position sur la Terre. Le récepteur va pouvoir affiner sa position grâce au décalage entre l'heure de départ du récepteur GPS et l'heure exacte sur l'horloge atomique du quatrième satellite.

Cependant, le signal transmis par les satellites est relativement faible et donc il peut être dégradé par la présence de bâtiments urbains, de montagnes ou encore de forêts. C'est ce que l'on appelle l'effet *tunnel*. En effectuant des tests, nous avons pu nous rendre compte que le point de déclenchement du message à l'endroit voulu était dur à trouver pour un faible rayon choisi (par exemple cinq mètres).

2.2.2 Résolution du problème de l'effet tunnel

C'est finalement ce problème que nous avons rencontré lors de la création du serveur dédié à la balade sonore. Etant donné que le parcours se situe en ville, et

entre de grands bâtiments, il a fallu prendre en compte les erreurs d'imprécisions de géolocalisation. Ainsi, nous avons mis un rayon de déclenchement suffisamment élevé afin que le son puisse être diffusé au bon endroit (vingt-cinq mètres).

Si vous souhaitez découvrir les rues de Clermont-Ferrand de manières plus ludiques, vous pouvez ajouter ce serveur au *Pigeon Nelson* :

https://lepigeonnelson.jmfavreau.info/Balade_Sonore/BaladeSonore.json
Le parcours est défini selon la carte (voir figure 4).

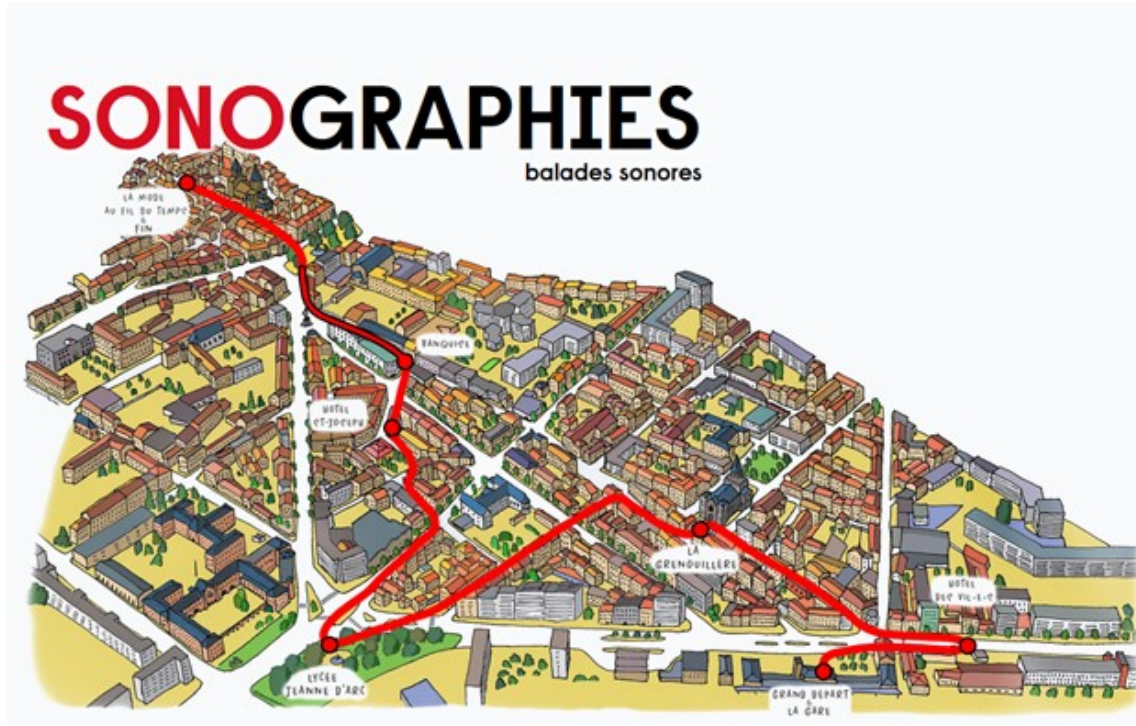


FIGURE 4 – Carte de la balade sonore autour de la gare de Clermont-Ferrand

3 Les serveurs dynamiques

Un serveur dynamique, quant à lui, va exécuter du codes côté serveur. Des données, généralement stockées dans des bases de données sur un serveur, seront envoyées côté client pour être affichées grâce à du code.

3.1 *OpenStreetMap*

3.1.1 Description

Afin de comprendre au mieux ce concept, nous avons exploité l'outil *OpenStreetMap* qui est une base de données géographique, libre de droit. On peut la partager, l'utiliser et la ré-utiliser comme on le souhaite. C'est un projet collaboratif à la disposition de tous, dans l'esprit Wikipédia, mais dédié à la cartographie. *OpenStreetMap* propose des données géographiques généreuses sur diverses thématiques tel que les

réseaux routiers, les bâtiments, les commerces ou points d'intérêts, etc. . .

Sa structure se compose essentiellement de formes géométriques et d'attributs. En effet, il s'agit en réalité d'un dessin composé d'objets complexes dont on va préciser la nature à partir d'attributs composé d'un système de clé/valeur. Chacun peut y contribuer, et de nombreux objets restent à l'heure d'aujourd'hui non répertoriés.

3.1.2 Contribution

Nous y avons contribué en rajoutant certains éléments comme le basic-fit de Brive-La-Gaillarde ou encore le lavoir d'un petit village perdu. Pour ajouter un bâtiment, il est plus facile d'utiliser un fond de carte cadastral (**voir figure 5**). Ensuite, il faut utiliser l'outil polygone (**voir figure 5**), qui permet de tracer de manière précise les contours du bâtiment. De plus, il est possible de rendre la forme orthogonale via des paramètres. Enfin, il est nécessaire de choisir le type de bâtiment comme une maison par exemple, et on peut renseigner des informations supplémentaires tels que l'adresse, le nombre de niveau, etc. . .

Si vous souhaitez apporter votre aide vous aussi, veuillez cliquer directement ici : [lien contribution](#).

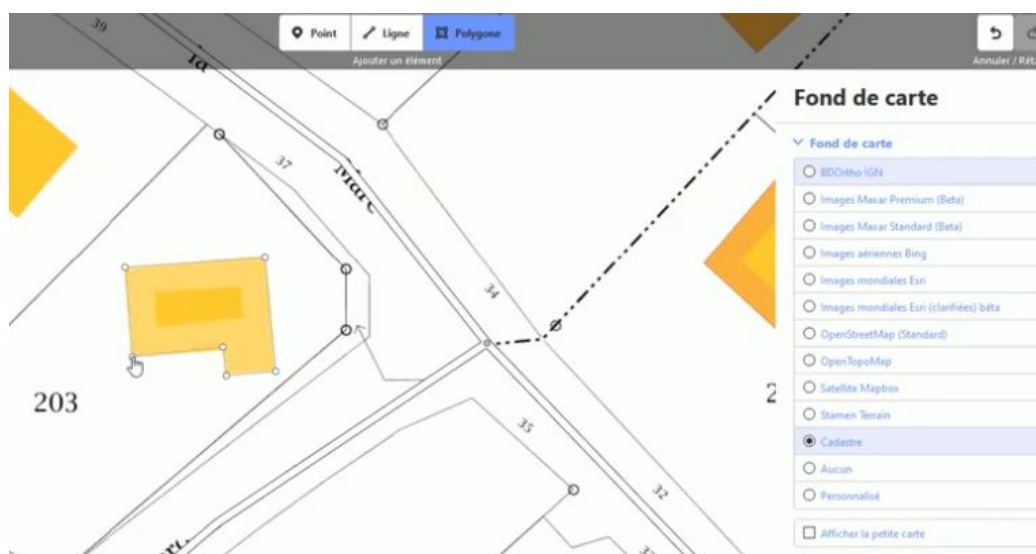


FIGURE 5 – Ajout d'une maison sur *OpenStreetMap*

3.2 *OverPass Turbo*

Afin de manipuler ces différentes données, nous pouvons utiliser *OverPass Turbo* qui est un outil permettant d'extraire et d'exploiter les données d'*OpenStreetMap*. Il suffit d'exécuter des requêtes et le résultat sera représenté sous forme d'une carte interactive (**voir figure 6**).


```
// possibilité de récupérer du json en
// sortie
[out:json][timeout:25];

(
  // Boulangerie à moins de 500m de la
  // place Jaude
  node["shop"="bakery"] (around:500,
    {{geocodeCoords:Place Jaude Clermont}}) ;
  way["shop"="bakery"] (around:500,
    {{geocodeCoords:Place Jaude Clermont}}) ;
  relation["shop"="bakery"] (around:500,
    {{geocodeCoords:Place Jaude Clermont}}) ;
);

// affichage du résultat
out body;
>;
out skel qt;
```

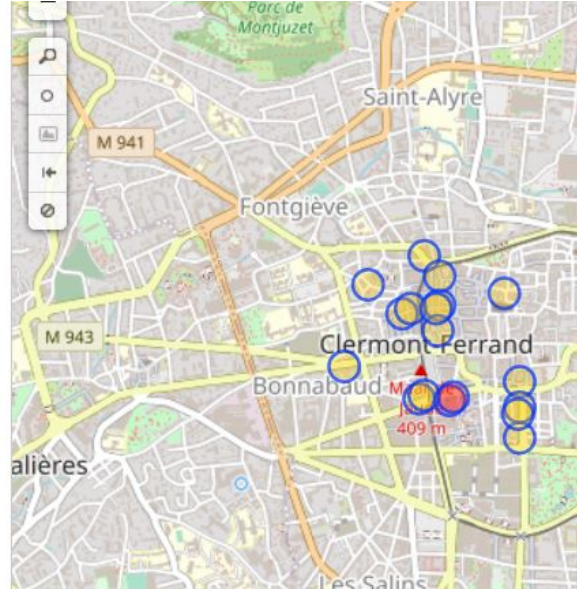


FIGURE 6 – Recherche des boulangeries à moins de cinq cents mètres de la place Jaude à partir de *OverPass*

3.2.1 Les différents paramètres

Le paramètre [timeout :25] fixe le temps maximal (en seconde, ici vingt-cinq) pour que la requête soit exécutée, et si ce temps est dépassé cette dernière sera annulée.

Les paramètres suivants concernent la zone dans laquelle chercher un élément répertorié sur *OpenStreetMap* :

- {{bbox }} : requête par rapport à la boîte englobante de la fenêtre de la carte courante
- {{center}} : requête par rapport au centre de la fenêtre de la carte courante
- {{geocodeId : name}} : requête par rapport à l'identifiant *OpenStreetMap* d'un lieu
- {{geocodeArea : name}} : requête par rapport à l'identifiant *OverPass* – *Area* d'un lieu
- {{geocodeCoords : name}} : requête par rapport à des coordonnées GPS d'un lieu (ex : geocodeCoords : Place Jaude Clermont => cela permet d'effectuer la requête par rapport aux coordonnées de la Place Jaude (**voir figure 6**)).

Il est également possible de choisir un rayon de recherche autour de la zone en question 'around : rayon' dont le rayon sera défini en mètres (**voir figure 6**). Pour définir l'élément que nous recherchons, il existe plusieurs types :

- node : un point (un nœud)
- way : deux nœuds reliés entre eux donc un chemin
- relation : plusieurs nœud reliés entre eux donc un objet complexe (ex : bâtiment)

Lors d'une requête *OverPass*, il est préférable d'utiliser les trois types car nous ne savons pas à partir duquel a été créé l'élément que nous recherchons. Dans chaque type, il faut spécifier un attribut clé/valeur qui est documentés dans le wiki (lien du wiki). Vous y trouverez une zone de recherche (voir figure 7 en rouge) et son résultat sous forme clé/valeur vous sera donné dans la page (voir figure 7 en vert). Ainsi, si vous recherchez une boulangerie ou encore un musée, leur attribut respectif seront 'shop'='bakery' et 'tourism'='museum'.



FIGURE 7 – Utilisation du wiki pour effectuer une requête *OverPass*

3.3 *OpenStreetMap* et *OverPass* au service du *Pigeon Nelson*

Ces outils vont nous être très utiles pour la création de certains serveurs. Par exemple, une requête *OverPass* a été utilisée dans le serveur public *Musées*. Ce serveur, codé en PHP, va grâce à la méthode *getOSData* (de la classe *PigeonNelsonServer*), prenant en paramètres une requête *OverPass* et une distance, stocker les différents musées présents dans la zone (à savoir deux kilomètres de rayon).

```
$server->getOSMData( '[ out:json ][ timeout:25 ];
(node[ "tourism"="museum" ]( { { box } } );
way[ "tourism"="museum" ]( { { box } } ); out center; ',
$radiusSearch );
```

Cette méthode va décoder un fichier JSON, généré grâce à la requête *OverPass*, contenant les différents musées qui seront stockées dans : *\$server*, et par la suite des messages pour chacun seront créés. À partir de ces données, le *Pigeon Nelson* va jouer du son si un musée se situe à moins de cinq cents mètres de l'utilisateur. Si le nom du musée a pu être récupéré à partir de la requête *OverPass* alors le message sera : "vous êtes proche d'un musée qui s'appelle *nom du musée*". À l'inverse, si aucun nom n'a pu être trouvé alors le message sera : "vous êtes proche d'un musée dont on ne connaît pas le nom". Quoi qu'il en soit, chacun des messages est gardé en mémoire et le *Pigeon Nelson* jouera le son si l'utilisateur se rapproche et se trouve dans un rayon de cinq cents mètres autour d'un musée.

4 Notre contribution au *Pigeon Nelson*

Nous avons contribué au *Pigeon Nelson* en créant un serveur capable de compter des éléments (bancs, passages piétons, arbres, ...) dans un rayon choisi autour de nous. Pour le réaliser, il y a eu plusieurs étapes progressives qui nous ont finalement permis d'aboutir à la version finale.

4.1 Les étapes

4.1.1 Première étape

Lors de la première étape nous avons simplement commencé par créer un serveur permettant de compter le nombre d'arbres dans un rayon de cinq cents mètres autour de nous. Il a fallu fabriquer une requête *OverPass* et récupérer les données de la même manière que pour le serveur musée. Ensuite, nous avons compté chacun des éléments stockés, ce qui est revenu à compter le nombre d'arbre.

4.1.2 Deuxième étape

La deuxième étape, quant à elle, a été de compter des éléments spécifiques dans un rayon choisi. Pour cela, nous avons dû créer un formulaire dont il a fallu par la suite récupérer les informations enregistrées. Nous avons affiché à l'utilisateur l'adresse du serveur jouant le message afin qu'il puisse l'entrer manuellement dans le *Pigeon*. Ainsi, selon l'élément récupéré et le rayon donné, une requête *OverPass* a été créée puis utilisée de la même manière qu'à l'étape précédente.

4.1.3 Troisième étape

La dernière étape a été de proposer et de mettre en œuvre quelques améliorations. Tout d'abord, nous avons rajouté un identifiant à l'adresse du serveur permettant de jouer le message. Cela a permis, à ce que plusieurs utilisateurs puissent remplir le formulaire en même temps mais avec des champs différents. Ensuite, nous avons apporté d'autres possibilités à l'utilisateur d'ajouter ce serveur dans le *Pigeon Nelson*. Il est à présent possible de flasher un QR Code ou alors de directement cliquer sur le lien ce qui ajoute automatiquement le serveur dans le *Pigeon*. Une autre amélioration a été de rendre le nom de ce serveur ajustable en fonction de la demande de l'utilisateur. Par exemple, si l'utilisateur voulait connaître le nombre de lampadaires dans un rayon de six cents mètres alors le nom du serveur est : "Nombre de lampadaires dans un rayon de 600 mètres." (**voir figure 8**). Enfin, nous nous sommes penchés sur les erreurs dû à un mauvais remplissage du formulaire. Si un champ est manquant, alors nous proposons à l'utilisateur l'adresse du serveur le plus récent. Avant cela, une erreur était directement générée sur la page web.

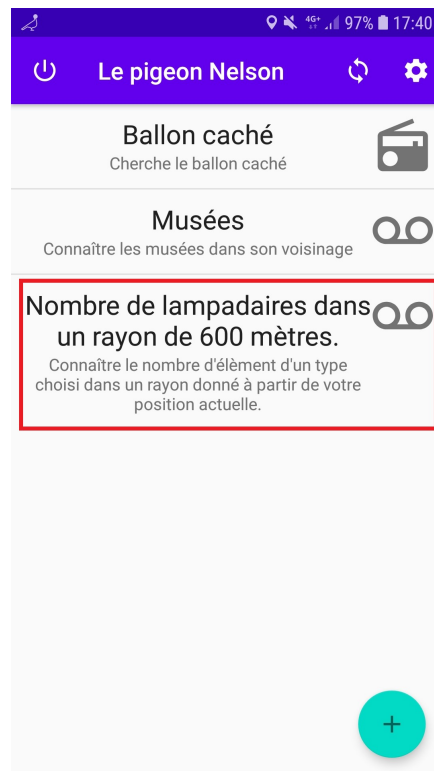


FIGURE 8 – Nom du serveur adaptable

4.2 Le fonctionnement du serveur

Ce serveur fonctionne grâce à trois fichiers principaux et un fichier JSON de la base de données.

Tout d'abord, il y a le fichier du formulaire (codé en HTML) qui permet à l'utilisateur de choisir un élément et un rayon (**voir figure 9**).

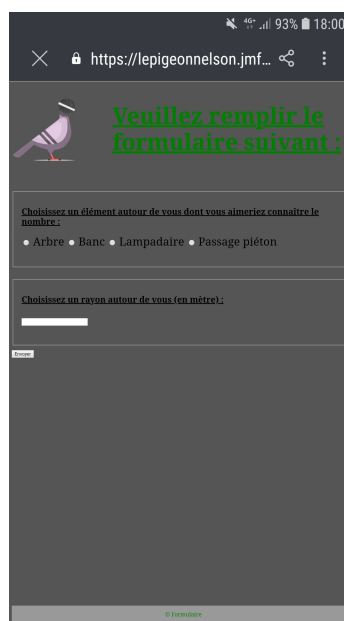


FIGURE 9 – Page web du formulaire

Ensuite, il y a un fichier qui, quant à lui, est codé principalement en PHP. Il va nous permettre de récupérer les informations enregistrées dans le formulaire. Une fois récupérées, nous allons vérifier que l'utilisateur ait bien rempli le formulaire, autrement dit, qu'il n'y ait aucun champ manquant.

Si c'est le cas, notre fichier va créer un autre fichier au format JSON qui aura comme clés 'element' et 'rayon' associés à leur valeur respective (valeur choisie par l'utilisateur au moment du remplissage du formulaire). Puis, une adresse correspondant au serveur jouant le message sera mise à disposition. C'est cette adresse qui doit donc être ajoutée dans le *Pigeon Nelson*. Pour cela, nous avons laissé deux possibilités à l'utilisateur, soit de flasher un QR Code ou soit de cliquer sur un lien ouvrant et entrant l'adresse automatiquement dans le *Pigeon* (**voir figure 10**).










FIGURE 10 – QR Code ou lien pour ajouter le serveur

De plus, cette adresse comporte un identifiant qui n'est rien d'autre que le numéro du fichier au format JSON correspondant. Afin de déterminer cet identifiant et donc de créer le bon fichier JSON, nous recherchons le fichier JSON qui a été créé le plus récemment et nous lui ajoutant un. Par exemple, si le fichier le plus récent est '*fichier44.json*' alors le nouveau fichier sera '*fichier45.json*'. Tout ces fichiers sont stockés dans le même emplacement (**voir figure 11**).

Dans le cas où l'utilisateur aurait mal rempli le formulaire alors l'adresse du serveur affichée sera celle dont l'identifiant correspond au fichier JSON le plus récent.

Index of /formulaire/fichierAudio

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 results1.json	2021-05-29 17:38	38	
 results2.json	2021-05-29 18:18	37	
 results3.json	2021-05-29 18:25	31	
 results4.json	2021-05-29 18:25	32	
 results5.json	2021-05-29 18:25	32	
 results6.json	2021-05-29 18:25	31	
 results7.json	2021-05-29 18:25	37	
 results8.json	2021-05-29 18:26	36	
 results9.json	2021-05-29 18:26	30	
 results10.json	2021-05-29 18:26	31	

Apache/2.4.38 (Debian) Server at le pigeonnelson.jmfavreau.info Port 443

FIGURE 11 – Emplacement des différents fichiers JSON

Enfin, le dernier fichier est lui aussi codé en PHP et c'est ce fichier qui correspondra au serveur à entré dans le *Pigeon*. Il permet de faire la requête *OverPass*, de compter le nombre d'éléments dans le rayon demandé et de donner la réponse par synthèse vocal. Grâce à l'identifiant de l'adresse comme vu précédemment, le fichier au format JSON pourra être retrouvé. Il comporte l'élément ainsi que le rayon choisi par l'utilisateur. Ainsi, la requête adéquate pourra être effectuée. Finalement, il ne restera plus qu'à compter le nombre d'éléments, s'il y en a ; puis à créer le message qui sera lu par synthèse vocale.

5 Conclusion

A la fin de ces quatre mois d'implication dans le projet, nous avons amélioré notre compréhension générale du web. De plus, nous avons contribué à la création d'un serveur pour le *Pigeon Nelson*, ce qui nous a permis de découvrir plein de concepts nouveaux tels que le langage PHP ou encore le format JSON. Bien que notre serveur final ne soit pas le plus optimisé, nous sommes contents du résultat car au tout début il nous était difficile d'imaginer pouvoir réussir à concevoir un serveur. C'est en avançant étape par étape, que nous sommes parvenus à comprendre et à progresser dans ce milieu.

Par rapport à notre serveur, nous avons d'autres idées d'amélioration. En effet, nous avons remarqué un potentiel problème en ce qui concerne notre solution en cas d'erreur dans le formulaire. Si un champ du formulaire est manquant, alors l'adresse sera donnée en fonction du fichier JSON le plus récent. Or, si aucun fichier JSON ne fait partie de la base de données alors cette solution ne fonctionnera pas. Une alternative possible à laquelle nous avons pensé est alors d'obliger les champs du formulaire à être remplis.

Par ailleurs, comme nous utilisons *OpenStreetMap* où tous les éléments existants ne sont pas forcément répertoriés, alors la réponse du *Pigeon* n'est pas tout à fait exacte.

Finalement, savoir combien de passages piétons se trouvent dans un rayon de trois kilomètres autour de nous n'est pas forcément très utile. Cependant, cela montre qu'avec un peu d'imagination et en s'inspirant de serveurs déjà existants, nombreuses sont les possibilités! Initialement, le *Pigeon Nelson* a été créé en tant qu'outil permettant de faciliter un projet complexe ayant pour but la description de carrefour pour déficient visuel.