Table des matières

Liste des tableaux

Liste des figures

Liste des abréviations

AJAX	Asynchronous JAvascript and XML
API	Application Programming Interface
BD	Base(s) de données
FIFO	First In First Out
GIS	Geographic Information System
GPS	Global Positioning System
HTML	HyperText Markup Language
JS	JavaScript
JSON	JavaScript Object Notation
OSM	OpenStreetMap
PHP	PHP : Hypertext Preprocessor
POI	Point of Interest
REST	REpresentational State Transfer
SIG	Système d'Information Géographique
SOSM	Swiss OpenStreetMap Association
XML	eXtensible Markup Language

Introduction

L'accessibilité des lieux publics est un problème essentiel pour les personnes à mobilité réduite. Il ne s'agit pas seulement de l'entrée dans le lieu qu'elles souhaitent fréquenter, mais aussi du trajet qui y mène ou de la présence de places de parking adaptées. Ce qui semble aisé pour la majorité des gens peut s'avérer très compliqué pour les personnes concernées par le handicap.

De nombreuses applications permettent de calculer un itinéraire porte à porte mais elles ne sont pas adaptées pour les fauteuils roulants, les personnes âgées et les familles avec poussettes. Il existe quelques applications, mais le plus souvent elles ne sont limitées qu'à une région particulière. De plus, les données concernant les lieux publics ne sont pas concentrées dans un seul emplacement mais doivent actuellement être récoltées dans plusieurs endroits par les personnes qui en ont besoin, par exemple en se renseignant auprès de chaque institution. Il serait utile d'avoir une solution qui structure et regroupe l'information actuellement dispersée et déstructurée.

En collaboration avec l'Institut du travail social et l'Institut de tourisme, l'Institut d'informatique de gestion de la HES-SO Valais-Wallis souhaite fournir une plateforme de collecte de données par le *crowdsourcing* et de génération d'itinéraire pour les personnes à mobilité réduite, WE-MAP. Elle est destinée avant tout aux personnes en fauteuil roulant, mais aussi aux personnes âgées, aux familles avec poussettes et, plus généralement, à toutes les personnes qui éprouvent des difficultés à se déplacer.

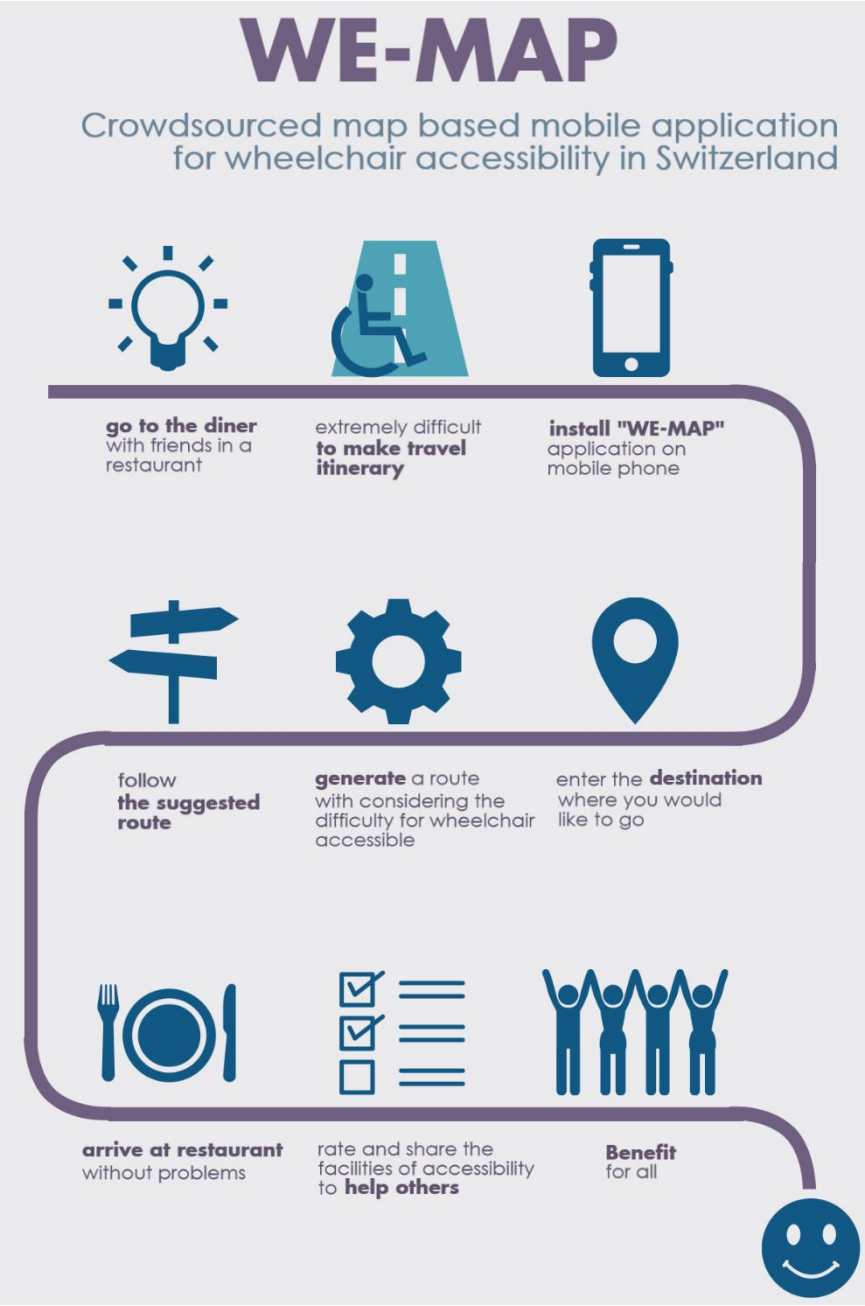
Dans sa version antérieure à notre travail, la plateforme permettait déjà la saisie de données d'une part sur les *points of interest* et d'autre part sur les routes. Ces données s'apparentent soit à des difficultés d'accès comme une pente de plus de 10°, un trottoir de moins d'un mètre cinquante ou un obstacle, soit à des services comme un accès sans marche, un parking ou des toilettes pour personnes en situation de handicap. Ces six critères ont été déterminés en collaboration avec l'Institut du travail social. Par contre, avant notre travail, l'application ne permettait pas encore le calcul d'itinéraires. Le projet WE-MAP dans son ensemble est planifié sur 18 mois.

Plusieurs centaines de données avaient déjà été saisies par les facteurs de Sierre, de Sion et du Val d'Anniviers. En effet, la grande force de l'application, c'est que les utilisateurs peuvent entrer eux-mêmes les informations au fur et à mesure qu'ils en prennent connaissance. Cela permet potentiellement une collecte d'informations presque infinie et très exhaustive. Cette sous-traitance d'un travail auprès de la foule des utilisateurs est appelée *crowdsourcing*. C'est

une force, mais c'est aussi un point délicat, car il faut motiver les utilisateurs à contribuer. L'application ne vaut rien si elle ne dispose pas de données fiables et exhaustives.

Un *use case* de WE-MAP, illustré par la Figure 1, pourrait être le suivant. Une personne en fauteuil roulant souhaite se rendre un soir dans un restaurant et s'inquiète de savoir comment elle va y aller. Après avoir installé l'application, elle saisit le parcours sur son téléphone et reçoit toutes les informations nécessaires pour son trajet. Arrivée au restaurant, qui n'est pas

Figure 1 – Un use case *de WE-MAP.*



Source : Liu, Glassey Balet, Sokhn, & De Gaspari (2017, p. 4)

encore référencé dans WE-MAP, elle saisit la présence de toilettes adaptées ou d'une entrée sans marches. Sa contribution sera utile à d'autres utilisateurs, qui sauront qu'ils peuvent sans problème se rendre dans ce restaurant (Liu, Glassey Balet, Sokhn, & De Gaspari, 2017, p. 4).

Ce travail de Bachelor a été principalement consacré au calcul de routes, en tenant compte des difficultés des personnes à mobilité réduite et à l'affichage du résultat. C'était là le cœur de notre responsabilité. Nous nous sommes cependant aussi penchés sur le *crowdsourcing*, la collecte de l'information étant déléguée à la foule (*crowd*) des utilisateurs, et sur les moyens de motiver ces utilisateurs. Nous avons enfin fait une évaluation intermédiaire de l'application.

Notre travail comportait quatre phases.

Dans la première phase, nous avons effectué un état de l'art en ce qui concerne le calcul d'itinéraires. Nous avons collecté quelques informations sur des applications existantes qui visent des buts semblables à WE-MAP. Puis nous nous sommes plongés dans les algorithmes de calcul d'itinéraires cités par la littérature. Il y a Dijkstra, bien sûr, mais aussi certains de ses dérivés comme A* ou les contractions hiérarchiques et encore Sedgewick et Vitter, Bellmann, Esopo et Pape, etc. Nous avons aussi étudié les outils disponibles pour notre travail : les systèmes de carte propriétaires, comme Google Maps, ViaMichelin et ArcGIS, ou *open source*, comme OpenStreetMap. Nous avons comparé leurs avantages et inconvénients respectifs pour déterminer si nous pouvions les utiliser dans notre application.

Dans une seconde phase, nous sommes passés au développement. L'application qui nous a été fournie était en HTML5, en PHP et en JavaScript, avec une base de données en MySQL. Elle utilisait la librairie JQuery et le framework JQuery mobile. Ce dernier est adapté pour faire des sites web-responsives et des applications accessibles aux smartphones. Elle se servait aussi de Google Maps JavaScript API pour l'affichage de la carte sur laquelle figurent les points d'intérêt. Nous avons utilisé le résultat de la première phase pour décider de quelle technologie nous servir pour ajouter le calcul d'itinéraires à l'application WE-MAP. Notre choix s'est porté sur Google Maps, en particulier parce que l'application existante utilisait déjà cette API et pour la qualité relative des cartes en comparaison avec, par exemple, OpenStreetMap. Nous avons aussi géré l'affichage sur la carte des points répertoriés dans la base de données, afin qu'ils ne s'affichent que s'ils sont le long de l'un des trajets et qu'ils n'apparaissent en rouge que s'ils sont pourvus d'une difficulté cochée par l'utilisateur. Dans le cas inverse, ils sont en vert. Nous avons aussi amené dans cette phase quelques modifications à l'application existante sans rapport avec le calcul d'itinéraires à la demande des responsables du projet.

La troisième phase a consisté à établir un état de l'art du *crowdsourcing* et des moyens de parvenir à motiver la foule (*crowd*) des utilisateurs à contribuer. Nous avons donné des

exemples de participation involontaire, de motivation intrinsèque et de motivation extrinsèque. La motivation intrinsèque est liée à l'acte de participer lui-même, quand la motivation extrinsèque est liée à une rémunération ou à une autre récompense, non financière. Nous en avons tiré quelques conclusions pour WE-MAP.

Enfin, dans une quatrième phase, nous avons pratiqué une analyse qualitative, restreinte quant à l'échantillon d'utilisateurs, afin d'avoir une évaluation des fonctionnalités existantes et de l'utilité de l'application. Pour cela, nous avons choisi de faire des *focus groups*, des groupes de discussion. Des testeurs ont été priés d'expérimenter l'application durant une semaine, aidés d'instructions afin de parcourir toutes ses possibilités. Puis nous les avons réunis en groupes de trois ou quatre personnes pour débattre sur l'application, son utilisabilité, son utilité et le *crowdsourcing*, dans une discussion organisée et enregistrée. Cela nous a permis d'avoir l'avis de personnes qui n'étaient pas impliquées dans le projet. Un certain nombre de remarques reçues à cette occasion nous a permis d'apporter quelques modifications à l'application. D'autres idées ont été transmises aux responsables du projet.

1. État de l'art de la recherche d'itinéraire porte à porte

1.1. Présentation du problème

L'Institut d'informatique de gestion à Sierre a déjà développé dans le cadre du projet WE-MAP une application qui permet d'évaluer les points d'intérêt, en anglais *points of interest* (POI), de Google Maps selon six critères, la présence de places de parc et de toilettes pour handicapés, l'absence de marches à l'entrée, une pente trop raide, un trottoir trop étroit ou un autre obstacle physique empêchant le passage. Les chemins et routes peuvent aussi être évalués selon ces trois derniers critères, soit la pente trop raide, le trottoir trop étroit ou l'obstacle physique. Ces critères figurent dans le Tableau 1.

Tableau 1 – Critères d'accessibilité et leurs définitions

Nom du critère	Définition
Parking pour handicapés	Places identifiées et réservées aux personnes handicapées
Toilettes pour handicapés	Toilettes adaptées pour les handicapés, incluant les exigences quant à la taille du local, un espace large et une poignée
Entrée sans marches	L'entrée des bâtiments ne comporte pas de marches
Pente > 10 degrés	Pente de plus de 10 degrés d'angle
Obstacles sur le chemin	Certaines formes de substrats sont un obstacle à l'accessibilité (trop mou, gravier ou herbe)
Trottoir < 1.5 m	Il faut une surface d'au moins 1.5m x 1.5m pour permettre à l'utilisateur d'une chaise roulante de s'arrêter pour se reposer sans gêner le flux des piétons

Source : d'après Liu, Glassey Balet, Sokhn, & De Gaspari (2017), p. 8.

Nous souhaitons apporter à l'application existante la possibilité pour les personnes à mobilité réduite de calculer le déplacement d'un endroit à un autre sur la carte, en évitant ce qui pour elles serait un obstacle, parmi ces trois derniers critères des chemins et routes.

Le problème de la recherche d'un itinéraire porte à porte est aussi appelé problème du plus court chemin. Ce calcul de la route optimale est un modèle d'application d'algorithmes dans le monde réel (Sanders & Schultes, 2007, p. 23).

Il existe actuellement de nombreux systèmes de navigation qui permettent de trouver son chemin à pied. La difficulté particulière de ce travail réside dans la possibilité pour l'utilisateur d'ajouter au calcul les obstacles qu'il souhaite éviter.

1.2. Applications semblables

Nous avons trouvé quelques exemples d'applications semblables sur Internet. Elles sont le plus souvent limitées à une zone géographique restreinte.

1.2.1. Handimap.org

Le site français handimap.org est limité à l'Hexagone : Rennes, Montpellier et Lorient-agglomération. Pour ces trois villes et agglomération, il est possible de calculer un itinéraire d'un point à un autre pour différents types de handicap. Des services plus rudimentaires (position des places de stationnement adaptées, carrefours à feu sonores, lieux et organismes accessibles) sont offerts à La Rochelle et Nice. Une application disponible dans l'App Store d'Apple, Hérault-Mobility, offre le même genre de services pour Carnon-Plage et Balaruc-les-Bains. Cette application intègre une partie de *crowdsourcing*, qui « permet à tout usager inscrit sur Handimap de modifier et commenter l'accessibilité d'un tronçon de voirie ou d'un point d'intérêt (ERP, arrêt de bus, place de parking...) », selon les informations recueillies sur le site.

Figure 2 – Handimap.org : page d'accueil dédiée à la ville de Montpellier.

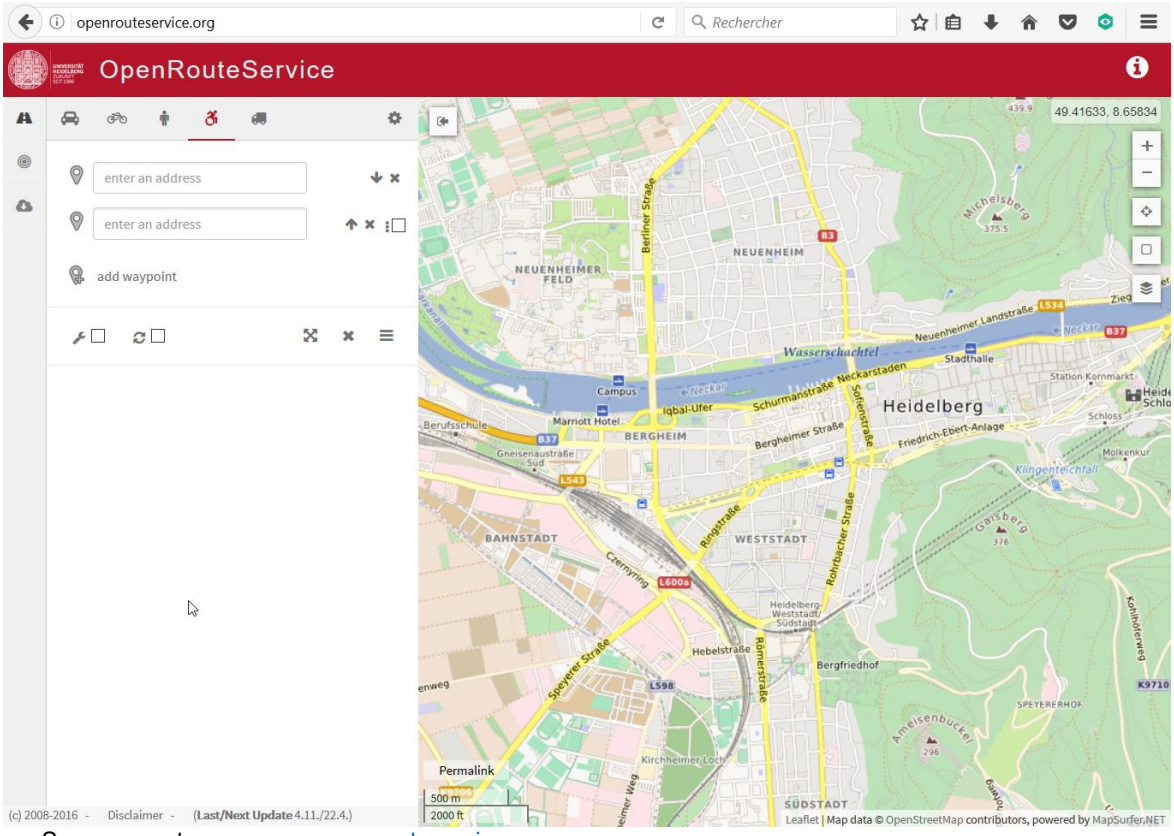
Source : www.handimap.org/Montpellier (Gervais & Morin, s.d.)

La Figure 2 montre, par exemple, la page permettant de calculer un itinéraire adapté pour les personnes souffrant de difficulté d'accessibilité motrice dans la ville de Montpellier (Gervais & Morin, s.d.).

1.2.2. OpenRouteService (Rollstuhlrouting)

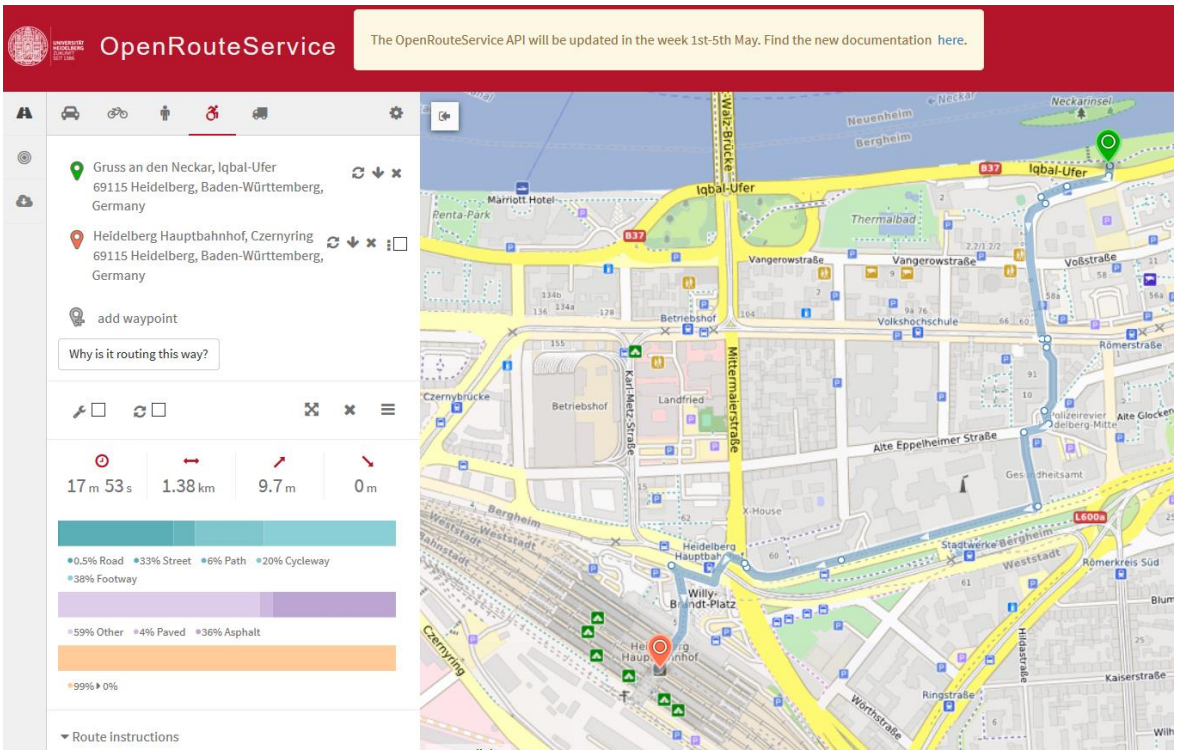
En Allemagne, l'Université de Heidelberg a mis au point un service accessible sur openrouteservice.org et sur rollstuhlrouting.de qui calcule théoriquement des itinéraires pour véhicules, vélos, piétons, handicapés et poids-lourds. La Figure 3 montre la page d'accueil de ce site Internet. Les bases de données sont fournies par OpenStreetMap (Universität Heidelberg, 2016). Il ne nous a cependant pas été possible de bien tester ce service. Il est en effet difficile de saisir des adresses précises ; il faut des adresses qui figurent dans la liste très restreinte fournie par le site. Même à Heidelberg, là où l'application aurait dû être testée, il n'y a que très peu de choix. La Figure 4 montre cependant un itinéraire de Iqbal-Ufer à la gare centrale de Heidelberg. Il est aussi possible de sélectionner un certain nombre d'options quant à la surface, la pente et la hauteur des seuils, comme le montre la Figure 5.

Figure 3 – OpenRouteService : service de calcul d'itinéraires en ligne créé par l'Université de Heidelberg, en Allemagne.



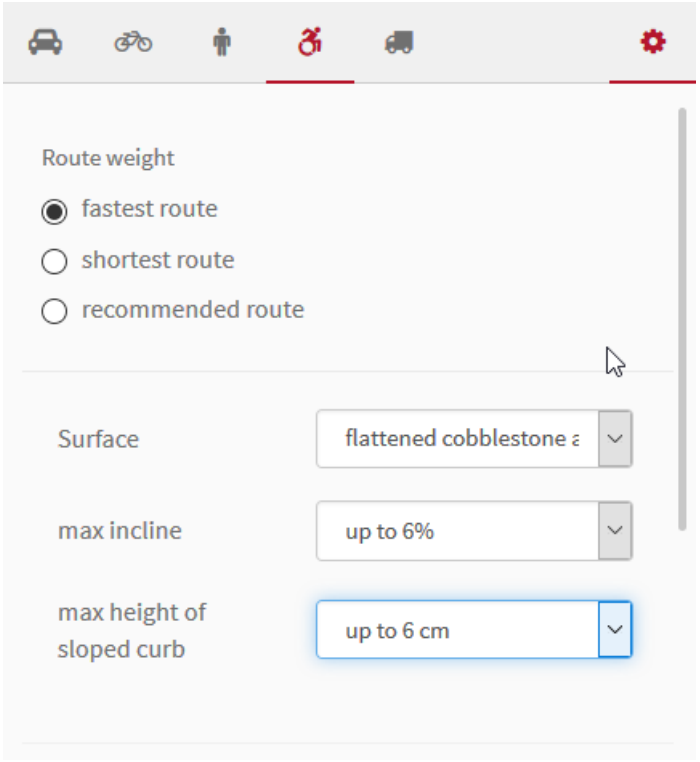
Source : capture sur www.openrouteservice.org.

Figure 4 – OpenRouteService (Rollstuhlrouting) : trajet en fauteuil roulant de Iqbal-Ufer à la gare centrale, à Heidelberg.



Source : capture sur www.rollstuhlrouting.de.

Figure 5 – OpenRouteService (Rollstuhlrouting) : options pour un parcours pour fauteuil roulant.



Source : capture sur www.rollstuhlrouting.de

1.2.3. Wheelmap.org

Le site wheelmap.org, fondé par l'association allemande Sozialhelden e. V., à Berlin, répertorie sur une carte, obtenue par les données de OpenStreetMap, les sites accessibles (vert), partiellement accessibles (orange) et sans accès (rouge) en fauteuil roulant. Les toilettes sont notées de même. Il n'y a, par contre, pas de calcul d'itinéraires (wheelmap.org, s.d.).

1.2.4. Synthèse des applications existantes

Les caractéristiques des applications citées dans ce chapitre 1.2 se retrouvent dans le Tableau 2. On voit que handimap.org se limite à certaines zones géographiques, villes ou agglomérations. Rollstuhlrouting ne donne aucune information sur l'accessibilité des points d'intérêt. Wheelmap, enfin, n'a aucune fonction de calcul d'itinéraire.

Tableau 2 – Caractéristiques de certaines applications existantes

	Handimap.org	OpenRouteService	Wheelmap.org
Crowdsourcing	✓	✓	✓
Zone géographique non limitée	✗	✓	✓
Données d'accessibilité	✓	✗	✓
Données sur place de parc	✓	✗	✗
Données sur les toilettes	✗	✗	✓
Calcul d'itinéraire	✓	✓	✗
Gestion d'obstacles	✓	✓	✗

Source : données de l'auteur

Notre application renseigne déjà sur les données d'accessibilité et sur la présence de places de parcs et de toilettes pour handicapés. Nous allons durant ce travail lui ajouter des fonctionnalités de calcul d'itinéraire, afin de la rendre plus complète que ce qui se fait déjà. De plus, nous ne nous limiterons pas à une zone géographique particulière.

1.3. Algorithmes de recherche d'itinéraire

On modélise le réseau de routes et chemins par un graphe pondéré dont les arcs sont les chemins qui peuvent être empruntés et les sommets les intersections. Nous définissons que le graphe comporte N sommets et M arcs. Nous avons trouvé dans la littérature plusieurs algorithmes qui permettent, à l'intérieur d'un graphe, d'aller d'un sommet source s à un sommet t en empruntant le plus court chemin. En réalité, la plupart des algorithmes, à l'exception de celui de Sedgewick et Vitter, calculent le plus court chemin du sommet s à tout autre sommet dans le graphe (Lacomme, Prins, & Sevaux, 2007, pp. 177-178).

La littérature distingue trois types de problèmes. Le problème A s'énonce « étant donnés deux sommets *s* et *t*, trouver un plus court chemin de *s* à *t*. Le problème B s'énonce « étant donné un sommet *s*, trouver un plus court chemin de *s* vers tout autre sommet ». Le problème C enfin consiste à trouver le plus court chemin entre tout couple de sommets ; il s'agit donc de calculer une matrice *N* x *N*, aussi appelée « distancier » (Lacomme, Prins, & Sevaux, 2007, p. 177) (Prins, 1996, p. 334). Dans ce projet, c'est le problème A que nous cherchons à résoudre. Or, nous verrons que bien des algorithmes donnent une solution au B. Cependant, quand le problème B est résolu, le problème A l'est aussi, car *t* est compris dans l'ensemble des sommets atteints dans le problème B.

Dans la suite de ce chapitre consacré aux algorithmes, nous allons utiliser une série de notations, inspirées de Lacomme, Prins & Sevaux, 2007, p. 181, qui figurent dans le Tableau 3. Pour un graphe donné, *N* désigne le nombre de sommets du graphe et *M* le nombre d'arcs. Pour deux sommets *i* et *j* reliés par un arc, $W_{(i,j)}$ désigne la valuation de cet arc, qu'on appelle aussi le poids ou le coût. Dans le cas d'un calcul d'itinéraires, il s'agira du temps de parcours entre ces deux sommets, voire éventuellement de la distance. Si le sommet source (ou point de départ) est *s*, alors pour tout sommet *x*, *V*[*x*] désigne la valeur du plus court chemin de *s* à *x* ; *V* est un vecteur. Un autre vecteur, *P*, contient les pères des sommets. Ainsi, *P*[*x*] est le père du sommet *x*. Par père, on entend le sommet par lequel on a atteint *x* dans un tracé optimal (il peut y avoir d'autres tracés optimaux, mais l'on n'en retient qu'un). En remontant d'un père à l'autre, on peut retracer en sens inverse le chemin optimal retenu.

Tableau 3 – Notations utilisées pour décrire les graphes et algorithmes dans ce travail.

Notation	Définition
N	Nombre de *sommets* du graphe considéré
M	Nombre d'arcs du graphe considéré
$W_{(i,j)}$	*Valuation* (ou *poids*, ou *coût*) de l'arc entre les sommets *i* et *j*
s	Sommet de départ (*source*)
***V*[*x*]**	Valeur du plus court chemin de *s* à *x*.
***P*[*x*]**	Père du sommet *x*, soit le sommet vers lequel il faut remonter en empruntant l'un des chemins optimaux.

Source : tableau de l'auteur, d'après Lacomme, Prins, & Sevaux (2007), p. 181.

Les algorithmes résolvant les problèmes de chemins optimaux sont répartis en deux familles, les algorithmes dits « à fixation d'étiquettes » (*label setting algorithms*) et les algorithmes « à correction d'étiquettes » (*label correcting settings*). Dans ce contexte, il faut comprendre par « étiquette » d'un sommet x la valeur totale du plus court chemin menant de la source à ce sommet x. Les algorithmes à fixation d'étiquettes traitent un sommet à chaque itération et calculent sa valeur de manière définitive. Au contraire, les algorithmes à correction

d'étiquettes peuvent affiner la valeur de chaque sommet jusqu'à la dernière itération (Lacomme, Prins, & Sevaux, 2007, p. 177).

La méthode qui pourrait sembler la plus directe consisterait à générer tous les arbres possibles entre tous les sommets du graphe, puis à calculer toutes les longueurs et garder le plus court. Mais dès que le nombre de sommets augmente, cela devient bien trop coûteux en calculs et irréalisable (Dijkstra, 1971, p. 64).

1.3.1. Algorithme de Dijkstra

L'algorithme de Dijkstra est un algorithme à fixation d'étiquettes ; à chaque itération, un des sommets reçoit son étiquette, sa valeur définitive. Il permet dans les faits de trouver le plus court chemin entre un sommet donné et tous les sommets atteignables depuis ce sommet (problème B). L'algorithme de Dijkstra n'est pas utilisable si certains arcs ont un poids négatif (Jungnickel, 2008, p. 76 ssq) (Lacomme, Prins, & Sevaux, 2007, pp. 182-183).

Le pseudo-code de la Figure 6 montre le fonctionnement de l'algorithme. On commence par initialiser chaque étiquette à l'infini ; dans le programme, on mettra une valeur assez haute pour être improbable. Seule l'étiquette du sommet source *s* est initialisée à 0. On crée un vecteur de booléens *Done*, initialisé à false ; on marquera dans la suite du programme comme *Done=true* les sommets traités définitivement ; il y en a un par itération. On entre ensuite dans l'algorithme, qui comporte une itération par sommet du graphe, si tous les sommets sont atteignables. On entre ensuite dans une boucle pour chercher le sommet *x* non marqué (*Done=false*) avec l'étiquette minimum. Si ce sommet *x* existe, on le marque (*Done=true*). On va alors boucler sur tous ses successeurs *y* pour mettre à jour leur étiquette, qui gardera le minimum entre sa valeur actuelle (*V*[*y*]) et la somme de l'étiquette de *x*, *V*[*x*], et du coût de l'arc *x*-*y*, *W*(*x*,*y*) ; en d'autres termes, si *y* a été atteint auparavant par un chemin plus court, il garde son étiquette, alors que si le chemin par *x* est plus court, c'est ce dernier qui est retenu (Geographic Information Technology Training Alliance (GITTA), 2016) (Jungnickel, 2008, pp. 76-77) (Lacomme, Prins, & Sevaux, 2007, p. 182).

Si tous les sommets du graphe sont accessibles, il y aura *N* itérations, soit une par sommet. Dans chaque itération, on cherche les successeurs du sommet traité. L'algorithme est donc en $O(N^2)$. Il est théoriquement possible, pour résoudre le problème A, de s'arrêter quand le sommet *t* est atteint ; mais cela ne diminue pas l'ordre de l'algorithme, car ce sommet pourrait, dans le pire des cas, être atteint en dernier (Lacomme, Prins, & Sevaux, 2007, pp. 182-188).

Figure 6 – Algorithme de Dijkstra : pseudo-code.

```
// Initialisations
Pour chaque sommet x :
        V[x] = ∞
        P[x] = null
        Done[x] = false // Done = valeur minimum de x atteinte

V[s] = 0
P[s] = s

// Algorithme
Minimum = 0
// Boucle sur tous les sommets atteignables
Tant que minimum ≠ ∞
        Minimum = ∞
        // Boucle : on cherche le sommet non marqué minimum x
        Pour chaque sommet v
                Si Done[v] = false && V[v] < Minimum
                        x = v
                        Minimum = V[v]

        // On traite ce sommet
        Si Minimum < ∞                  // x existe, on a trouvé un sommet
                Done[x] = true  // on marque x
                // Boucle : mise à jour des successeurs
                Pour chaque successeur y de x
                        V[y] = min( V[y], V[x] + W(x,y) )
```

Source : rédigé par l'auteur, d'après Geographic Information Technology Training Alliance (GITTA) (2016), Jungnickel (2008) pp. 76-77, et Lacomme, Prins, & Sevaux (2007) p. 182.

Dijkstra a comme désavantage qu'il n'est pas sensible à la densité du graphe. On peut utiliser un tas (*heap*) ce qui est avantageux pour un nombre d'arcs peu élevé (graphe peu dense). Le tas est initialisé avec *s* et contient à chaque itération les sommets non fixés de valeur non infinie. L'algorithme est alors en $O(M \cdot \log N)$. Si le graphe comporte tous les arcs possibles ($M = N^2$), c'est plus coûteux que la version de base sans le tas. Mais dans le cas qui nous occupe (un réseau de routes et de chemins), c'est loin d'être le cas (Lacomme, Prins, & Sevaux, 2007, pp. 186-188).

En plus du tas, une autre façon de rendre Dijkstra plus efficace est de l'appliquer de manière bidirectionnelle. L'idée de base est qu'on commence la recherche à la fois par la source et par la destination, soit en effectuant à tour de rôle une itération à chaque bout (avec chaque tas), soit en utilisant un seul tas dans lequel chaque élément sait à quel Dijkstra (depuis la source ou depuis la destination) il appartient. Cela permet de diviser seulement d'un facteur d'environ deux le calcul, mais cela en vaut la peine (Bast, 2012a).

1.3.2. Algorithme de Sedgewick et Vitter

L'algorithme de Sedgewick et Vitter est spécialisé dans la résolution du problème particulier du chemin entre deux sommets *s* et *t* donnés dans le graphe euclidien. Il fait une évaluation par défaut du coût B[*i*] d'un plus court chemin reliant les deux sommets (*s* et *t*) et passant par *i*. Ce coût se calcule comme suit :

$$B[i] = V[i] + D(i,t).$$

Cette dernière valeur désigne la distance euclidienne de *i* à *t*. Elle se calcule d'après les coordonnées X_i *et* Y_i de tout point *i* :

$$D(i,j) = \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2}.$$

Cet algorithme est aussi en $O(M \cdot \log N)$. Il n'est pas utilisable pour le cas qui nous occupe, car il utilise les distances à vol d'oiseau entre les points. Or, les tracés des chemins et routes ne sont pas construits ainsi dans la pratique (Lacomme, Prins, & Sevaux, 2007, pp. 188-189) (Sedgewick & Vitter, 1984, pp. 417-418).

1.3.3. Algorithmes à buckets

Les algorithmes à buckets sont des variantes de l'algorithme de Dijkstra. Il faut que les coûts des arcs soient des nombres entiers et que leur maximum ne soit pas trop grand. Il s'agit de partitionner l'intervalle des valeurs des étiquettes en *B* intervalles de largeur commune *L*. Ces intervalles sont numérotés de 0 à *B*-1. Les sommets qui y sont groupés en fonction de la valeur de leur étiquette forment les buckets. Les buckets sont stockés dans un vecteur de listes *Buck*. Ainsi, *Buck*[*k*] contient les sommets d'étiquette entre $k \cdot L$ et $(k+1) \cdot L - 1$.

L'algorithme fonctionne ensuite de la manière suivante : on cherche le *bucket* de plus petit indice *k* non vide. Dans ce *bucket*, on cherche par balayage le sommet *x* d'étiquette minimale. Pour chacun des successeurs *y* de *x*, on cherche son *bucket* (*Buck*[*V*[*y*]] / *L*), qu'on balaie pour y ôter *y*. Puis, on modifie l'étiquette de *y* comme avec Dijkstra : *V*[*y*] = *V*[*x*] + *W*(*x*,*y*). On localise alors le nouveau *bucket* de *y* (*Buck*[*V*[*y*] / *L*]) et on y insère *y*, en tête du *bucket*. Les sommets

étant en pratique bien répartis et les listes courtes, les performances moyennes sont bonnes (Lacomme, Prins, & Sevaux, 2007, p. 189).

1.3.4. Algorithme A*

L'algorithme A* – qui se prononce « a étoile » ou en version anglaise « a star » – est une extension de l'algorithme de Dijkstra. Il est largement utilisé dans la recherche du plus court chemin entre deux points. Il est reconnu pour sa performance et sa précision. Son nom sibyllin mérite une explication. En 1964, Nils Nilsson inventa une approche heuristique pour améliorer l'algorithme de Dijkstra. Il la nomma A1. En 1967, Bertram Raphael l'améliora de manière remarquable mais sans réussir à prouver ce progrès. Il appela cet algorithme A2. Enfin, en 1968, Peter E. Hart put prouver non seulement qu'il était plus efficace, mais même qu'il était le meilleur algorithme possible dans les conditions données. Il le nomma donc A* selon la syntaxe de Kleene, car c'est l'algorithme commençant par A et incluant toutes les numéros de versions possibles (Nosrati, Karimi, & Hasanvand, 2012, p. 251).

A* utilise une fonction heuristique du coût total pour déterminer dans quel ordre les sommets vont être visités. Cette fonction nommée $f(x)$ se calcule ainsi pour un sommet *x* :

$$f(x) = g(x) + h(x)$$

où $g(x)$ est le coût exact depuis le sommet source *s* jusqu'au sommet courant *x* et $h(x)$ est une estimation heuristique admissible du coût depuis le sommet *x* jusqu'à la destination (Nosrati, Karimi, & Hasanvand, 2012, pp. 251-252).

Le premier membre de l'addition, $g(x)$, est assez facile à déterminer, car c'est la somme des arcs empruntés pour arriver à *x* en partant de *s*.

Le second membre, $h(x)$, est plus difficile à calculer. Ce doit être une heuristique admissible. Il ne doit pas surestimer la distance, car alors le résultat serait faux. Il faut donc une estimation optimiste de la plus petite distance physique possible entre le sommet *x* et la destination. Pour une application de calcul d'itinéraire, $h(x)$ représente la distance à vol d'oiseau vers le but. Pour certains jeux à cases, ce sera la distance de Manhattan (Nosrati, Karimi, & Hasanvand, 2012, p. 252).

Cet algorithme a pour avantage sur Dijkstra qu'on n'explore pas tous les sommets accessibles, ce qui permet de gagner du temps. Par contre, comme il repose sur une heuristique, on n'a pas la garantie d'avoir le meilleur chemin possible. Le premier chemin trouvé sera un des meilleurs possibles.

L'algorithme utilise deux listes de sommets, comme l'illustre le pseudo-code de la Figure 7. La première liste, *Open*, contient les sommets qui ont été visités mais dont les successeurs n'ont pas été explorés. C'est la liste des tâches en suspens. La seconde liste, *Closed*, contient les sommets qui ont été visités et dont les successeurs l'ont été aussi et inclus dans la liste *Open*. Au début, la liste Open ne contient que le sommet source *s*, dont on calcule la distance heuristique à la destination $h(s)$. Pour la source, $f(s) = h(s)$, car $g(s)$ est nul. On boucle ensuite sur les sommets visités. On prend le sommet *x* ayant le plus petit $f(s)$, on le retire d'Open pour le mettre dans Closed. On boucle ensuite sur chacun de ses successeurs.

Figure 7 – Algorithme A : pseudo-code.*

```
// Initialisations
s = sommet source
t = sommet destination
liste Open ∋ s
liste Closed vide
g(s) = 0
h(s) à calculer
f(s) = h(s)

// Algorithme
// Boucle sur les sommets visités
Tant que (Open n'est pas vide)
        Ôter x de Open (élément avec le plus petit f(x))
        Ajouter x à Closed

        // Boucle sur les successeurs de x
        Pour chaque successeur y de x
                Si y = t
                        On a trouvé le chemin, on peut s'arrêter là
                Calculer son coût = g(x) + W(x, y)
                // 1re option : y est déjà dans Open
                Si y ∈ Open && coût < g(y)
                        g(y) = coût
                        P(y) = x
                // 2e option : y est déjà dans Closed
                Sinon si y ∈ Closed && coût < g(y)
                        g(y) = coût
                        P(y) = x
                        Ôter y de Closed et le mettre dans Open

                // 3e option : y n'est dans aucune liste
                Sinon si y ∉ Open && y ∉ Closed
                        g(y) = coût
                        P(y) = x
                        Calculer h(y)
                        Mettre y dans Open
```

Source : rédigé par l'auteur d'après Alsedà (2017) et Patel (2017).

Si l'un d'eux est le sommet destination *t*, on a trouvé un chemin et l'on s'arrête là. Sinon, on calcule le *coût* de *y* en passant par *x* ; il est de *g(x)*, le coût pour atteindre *x*, additionné du coût de l'arc de *x* à *y*, $W(x,y)$. Puis il y a trois cas :

Soit *y* est déjà dans Open, ce qui signifie qu'un autre de ces prédécesseurs a déjà été visité. Dans ce cas, si le chemin par x est plus avantageux que le précédent chemin qui a permis de le placer là, on l'y laisse, mais en adaptant ses $g(y)$ et $P(y)$ afin qu'ils correspondent au passage par x.

Soit *y* est déjà dans Closed, ce qui signifie qu'il a déjà été visité. Là aussi, si le chemin par *x* est plus avantageux que le précédent chemin qui a mené à *y*, on adapte ses $g(y)$ et $P(y)$; il faudra aussi visiter à nouveau *y*, c'est pourquoi on l'ôte de Closed pour le mettre dans Open.

Soit *y* n'est ni dans Open ni dans Closed : ni lui, ni ses prédécesseurs n'ont encore été visités. Le coût $g(y)$ pour atteindre *y* est donc celui qu'on a calculé, son père est *x* ($P(y) = x$). On calcule $h(y)$, le coût heuristique pour aller de *y* à la destination. On place enfin *y* dans la liste Open des sommets à visiter (Alsedà, 2017) (Patel, 2017).

Comme pour Dijkstra, on peut diviser d'un facteur deux environ la recherche en commençant par les deux côtés à la fois, la source et la destination (Bast, 2012a).

1.3.5. Contractions hiérarchiques

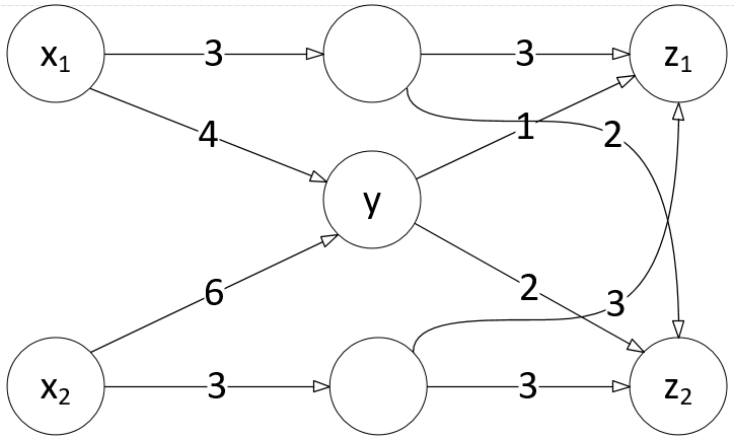
Le terme anglais *Contraction Hierarchies*, qui signifie littéralement « hiérarchies de contraction » est traduit en français par « contractions hiérarchiques » sur Wikipedia et d'autres sources (Bonifas, 2013).

La méthode des contractions hiérarchiques est un algorithme dérivé de Dijkstra proposé par des chercheurs de l'Institut d'informatique théorique à Karlsruhe. Il s'agit, pour des graphes qui, comme un réseau routier, peuvent être très grands avec des millions de nœuds, de contracter le graphe. Pour contracter le graphe, on utilise une heuristique assez intuitive. Quand on s'éloigne des sommets source et destination, on ne considère que les arcs de grande importance. Typiquement, pour un réseau routier, les autoroutes sont de première catégorie, puis les routes principales, puis les secondaires. Si on est à une distance plus grande qu'une valeur déterminée des source et destination, on ne considère que les autoroutes. À une distance intermédiaire, les routes principales. Plus près, toutes les routes. Les sommets intermédiaires sont provisoirement ôtés du graphe. Mais il faut remplacer une partie des arcs qui y arrivent et en partent. Pour un de ces sommets *y* supprimé, si x_i sont les sommets parents et z_i les sommets enfants, pour chaque paire d'arcs (x_i, *y*), (*y*, z_i), si le chemin (x_i, *y*, z_i) est le plus court chemin entre x_i et z_i, on ajoute un arc (x_i, z_i) de poids

$W(x_i, z_i) = W(x_i, y) + W(y, z_i)$. On applique ensuite Dijkstra sur le graphe simplifié, en bidirectionnel, c'est-à-dire en commençant en même temps de la source et de la destination. (Bast, 2012a) (Geisberger, 2008) (Geisberger, Sanders, Schultes, & Delling, 2008) (Bast, 2012b).

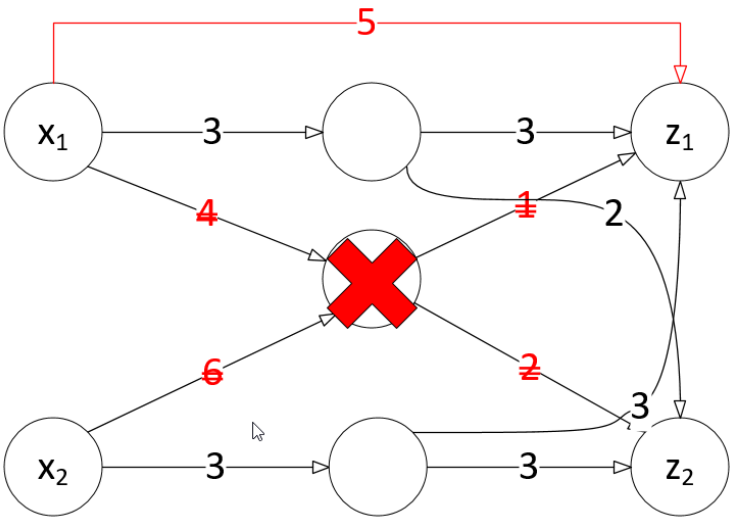
La Figure 8 montre un graphe avant contraction. Si on contracte le sommet *y*, comme le plus court chemin de x_1 à z_i passait par y (4 + 1 = 5), on doit ajouter un arc entre x_1 et z_1 de valeur 5, en rouge sur la figure. Par contre, les plus courts chemins entre x_1 et z_2, entre x_2 et z_2 et entre x_2 et z_1 ne passent pas par y. Il ne faut donc pas ajouter d'arc entre ces trois paires de points. La Figure 9 montre le graphe après contraction.

Figure 8 – Contraction hiérarchique : graphe avant la contraction.



Source : illustration de l'auteur.

Figure 9 – Contraction hiérarchique : graphe après contraction.



Source : illustration de l'auteur.

Si cet algorithme repose sur une idée assez simple, il est assez compliqué à implémenter sans erreur dans la pratique (Bast, Efficient Route Planning, SS 2012, Lecture 6 [Microsoft PowerPoint], 2012a, p. 11).

Dans la situation qui nous occupe, nous n'aurons en général pas un graphe immense, car les déplacements en fauteuil roulant se font le plus souvent dans un secteur relativement restreint. Nous n'aurions donc pas forcément besoin des contractions hiérarchiques. Mais ce n'est pas le cas des applications qui font du calcul d'itinéraires avec des réseaux routiers complexes qui forment des graphes démesurés.

1.3.6. Algorithme de Bellmann

Parmi les algorithmes à correction d'étiquettes, il y a l'algorithme de Bellman ou Bellman-Ford ou Bellman-Ford-Moore, selon les auteurs. Un des avantages de cet algorithme est qu'il autorise les poids négatifs des arcs, mais cela ne concerne évidemment pas notre cas. Il fonctionne par optimisation récursive selon la méthode décrite par les trois équations de la Figure 10, dans lesquelles il faut comprendre que $V_k(x)$ désigne la valeur du plus court chemin d'au maximum *k* arcs entre le sommet source *s* et le sommet *x*. La première de ces équations (a) indique que le sommet *s* est atteint en zéro arc pour un coût nul. La deuxième (b) que tout autre sommet *y* que *s* ne peut pas être atteint en zéro arc (le coût infini marque que ce n'est pas possible). Ces deux équations servent à stopper la récursion. La troisième équation (c) se comprend de la manière suivante. Pour chaque sommet *x* prédécesseur de *y*, le plus court chemin pour atteindre *y* en *k* arcs est égal à la partie entre accolades, soit le chemin le plus court pour atteindre ce *x* en *k*-1 arcs, additionné du chemin de *x* à *y*. On retient le minimum ainsi calculé (Lacomme, Prins, & Sevaux, 2007, pp. 194-195) (Jungnickel, 2008, pp. 80-81).

Figure 10 – Algorithme de Bellman: équations décrivant la méthode de programmation dynamique.

$$\begin{cases} V_0(s) = 0 & (a) \\ V_0(y) = \infty, y \neq s & (b) \\ V_k(y) = \min_{x \in \Gamma^{-1}(y)} \{V_{k-1}(x) + W(x,y)\}, k > 0 & (c) \end{cases}$$

Source : Lacomme, Prins, & Sevaux (2007) p. 194.

Figure 11 – Algorithme de Bellman : pseudo-code.

```
// Initialisation
Pour chaque sommet x
        V[x] = ∞
        P[x] = null

V[s] = 0
P[s] = s
k = 0
Stable = false

// Algorithme
// Boucle jusqu'à stabilité ou jusqu'à avoir fait N itérations
Tant que Stable = false et k < N
        k = k + 1
        // Réinitialisation de VNew
        Pour chaque sommet v
                VNew[x] = ∞

        // Booléen qui interrompra la boucle s'il n'y a plus de changement
        Stable = vrai

        // Boucle sur chaque sommet du graphe
        Pour chaque sommet y
                // Boucle sur chaque prédécesseur de y
                Pour chaque sommet x prédécesseur de y
                        Si (V[x]≠∞) && (V[x]+W(x,y) < VNew[y])
                                VNew[y] = V[x] + W(x,y)
                                P[y] = x
                                Stable = faux
                V = VNew
```

Source : adapté par l'auteur de Lacomme, Prins, & Sevaux (2007) p. 195.

La Figure 11 donne le pseudo-code de l'algorithme de Bellman. On commence par initialiser le vecteur *V* à infini pour toutes les étiquettes, sauf l'étiquette du sommet source qui est à zéro. Dans le programme, comme au chapitre 1.3.1, il faut choisir une valeur assez élevée pour n'être pas vraisemblable. Le vecteur des pères *P* est initialisé à *null*, sauf pour la source qui est son propre père. Le compteur k permettra de compter le nombre d'arcs et le booléen Stable sera faux tant qu'une amélioration est détectée dans l'algorithme. Pour gagner en

efficacité, on renonce à la récursivité dans le programme. On calcule le vecteur *V* itérativement. Pour chaque étape, on regarde si l'étiquette *V*[*y*] est améliorable en venant d'un prédécesseur de *y*. Puis on écrase *V* avec *VNew* et on passe à l'étape suivante. L'algorithme s'arrête quand *k* = *N* ; en effet, il ne peut pas y avoir plus de *N*-1 arcs, ce qui signifierait que le chemin est passé par tous les sommets. L'algorithme est en O(*M*·*N*), car il y au plus *N*-1 itérations principales, comme nous venons de le voir, et dans chaque itération on boucle sur chaque prédécesseur, dont il peut y avoir au pire un par arc, soit *M*.

1.3.7. Algorithme FIFO

L'algorithme FIFO est une forme dérivée de l'algorithme de Bellman. C'est donc aussi un algorithme à correction d'étiquettes. Il examine tous les sommets dans l'ordre First In First Out (FIFO). Pour cela, il utilise une file *Q*. Au début, seul *s* est dans la file. Il y a une itération par sommet dans la file. On prend le premier sommet de la file, on recherche ses successeurs et pour chacun d'entre eux, si leur étiquette est améliorée on les place en fin de file. On termine quand la file est vide. L'intérêt par rapport à Bellman est qu'on n'utilise pas les prédécesseurs. Il est aussi en O(*N*·*M*) (Lacomme, Prins, & Sevaux, 2007, p. 198).

1.3.8. Algorithme d'Esopo et Pape

L'algorithme d'Esopo et Pape, un autre algorithme à correction d'étiquettes, utilise une pile-file, soit une file dans laquelle il est possible d'ajouter un élément en tête comme en queue. Un sommet atteint pour la première fois est mis en fin de file, comme dans FIFO. Au contraire, si on l'atteint à nouveau par la suite, on l'insère en tête de file. L'idée sous-jacente est qu'un sommet atteint pour la première fois risque d'avoir des chemins mauvais dans un premier temps. Mais si on le visite à nouveau et qu'on a diminué son étiquette, il vaut la peine d'explorer immédiatement ses successeurs pour propager l'amélioration. Il peut avoir un comportement exponentiel en O(min(*N*·*M*·*U*, *M*·2^N) dans certaines configurations improbables en pratique. Il est cependant rapide en moyenne sur les graphes peu denses.

1.3.9. Algorithme de Floyd

L'algorithme de Floyd résout le problème C (cf. p. 10), c'est-à-dire qu'il calcule un distancier *N* x *N* des plus courts chemins entre chaque couple de sommets du graphe. Le vecteur des étiquettes devient une matrice *V*. Le plus court chemin de i à j est désigné par *V*[*i*,*j*]. La matrice initiale V^0 est initialisée selon :

$$\begin{cases} V_{ij}^0 = 0, si\ i = j \\ V_{ij}^0 = W(i,j), si\ (i,j)\ est\ un\ arc\ du\ graphe \\ V_{ij}^0 = \infty, si\ (i,j) n'est\ pas\ un\ arc\ du\ graphe \end{cases}$$

La première ligne désigne la diagonale de la matrice. Même s'il existe un arc-boucle allant d'un sommet à lui-même, cet arc n'a aucun effet sur le plus court chemin et peut être ignoré.

On calcule ensuite la matrice V^1, qui contient les plus courts chemins ayant le sommet 1 comme intermédiaire. Pour chaque arc (*i*,*j*) du graphe, on calcule $V_{ij}^1 = \min(V_{ij}^0, V_{i1}^0 + V_{1j}^0)$. On construit ainsi de suite *N* matrices V^k dont les éléments sont calculés selon $V_{ij}^k = \min(V_{ij}^{k-1}, V_{ik}^{k-1} + V_{kj}^{k-1})$. Il y a donc une boucle sur *k* et ensuite on boucle sur les deux dimensions de la matrice *i* et *j*. L'algorithme est donc en $O(N^3)$ et ne dépend pas du tout du nombre d'arcs (Lacomme, Prins, & Sevaux, 2007, p. 201).

1.3.10. Algorithmes utilisés par les applications

L'algorithme de Dijkstra est un très bon algorithme de recherche d'itinéraire, classique et bien éprouvé. Il trouvera à coup sûr le chemin le plus court d'un point A à un point B en explorant l'entier du graphe, quelle que soit la complexité de ce dernier. Cependant, dans le monde réel, le réseau routier est un graphe relativement simple de type planaire, c'est-à-dire que si on le dessine sur un papier les arcs (routes) ne se chevauchent pas. De plus, les applications existantes ne doivent pas forcément retourner le meilleur chemin, mais peuvent se contenter d'un chemin suffisamment court. Des systèmes tels que Google ou Viamichelin ne révèlent pas leur algorithme de calcul, qui fait partie de leurs secrets de fabrication. Il est vraisemblable d'estimer qu'ils n'utilisent pas simplement Dijkstra, mais plutôt des dérivés, en se servant d'heuristiques, comme le fait A*. Sans doutes utilisent-ils des heuristiques sophistiquées qui donnent un bon rapport entre la qualité de l'itinéraire calculé et l'efficacité du calcul (Golshan, 2015) (Byrne, 2015).

Les systèmes *open source*, quant à eux, indiquent souvent utiliser Dijkstra, A* ou les contractions hiérarchiques, parfois à choix pour le développeur. Le lecteur se reportera à la section 1.6 de ce document.

1.4. Sélection du graphe dans la base de données

Une des difficultés pour le calcul d'itinéraires est la taille importante du graphe. Par exemple, si nous voulons nous rendre en voiture de Sierre à Madrid, le nombre de tronçons de routes qui peuvent être empruntés est très grand. Le calcul d'itinéraires est donc vite assez lourd. En général, une personne en fauteuil roulant va parcourir des distances beaucoup moins importantes, relativement à un parcours en véhicule privé. Cela nous permet d'avoir un graphe beaucoup moins grand.

Quelles que soient les technologies utilisées, il faudra pouvoir restreindre le graphe à la région du point de départ et d'arrivée.

1.5. Systèmes de cartes existants

Il existe plusieurs systèmes de cartes et de calcul d'itinéraire. La plupart ne permettent pas d'exclure un des chemins.

1.5.1. Google Maps APIs

Google Maps offre toute une série d'Application Programming Interfaces (API) permettant de travailler sur les très grandes quantités de données réunies par cette organisation. Ces services sont gratuits jusqu'à un certain point. Ils nécessitent l'utilisation d'une clé d'application, délivrée par Google, qui permet de compter le nombre d'usages qui est fait de l'API (Google, 2017d).

Google Maps Directions API permet de calculer un itinéraire d'un point à un autre. Il est possible de spécifier dans la requête un mode de transport (paramètre *mode=*) qui peut être la voiture (*driving*), à pied (*walking*), en bicyclette (*bicycling*) ou en transports publics (*transit*). Rien n'est prévu pour les fauteuils roulants, mais c'est évidemment le mode *walking* qui s'en rapproche le plus. On peut définir un certain nombre de points de cheminement (*waypoints*), mais pas de point à éviter. Le paramètre *avoid* permet des restrictions d'itinéraires, mais il se limite à *tolls* pour les péages, *highways* pour les autoroutes, *ferries* pour les bacs et *indoor* pour éviter les trajets en intérieur lors de trajets à pied ou en transports en commun ; ce paramètre ne nous sera d'aucune utilité pour la restriction d'itinéraires pour personnes à mobilité réduite. Il est enfin possible de définir un paramètre *alternatives=true* qui aura pour effet que le service proposera plusieurs itinéraires (Google, 2017a). Cette API n'est donc pas vraiment idéale pour l'objet de notre travail. L'utilisation des *waypoints* pour contourner un obstacle n'est pas envisageable. Seul le calcul d'itinéraires alternatifs pourrait nous laisser espérer que l'un des itinéraires proposés évite de passer au mauvais endroit. Mais ce ne sera pas toujours une solution satisfaisante.

Ce service nécessite l'utilisation d'une clé d'application. Il est permis de faire 2'500 requêtes d'itinéraire gratuites par jour, comportant au maximum 23 points de cheminement et 50 requêtes par seconde au maximum. Il est possible, moyennant 0,50 USD pour 1'000 requêtes, d'augmenter le nombre de requêtes par jour jusqu'à 100'000. Au-delà, il faut adhérer au Google Maps APIs Premium Plan (Google, 2017c).

Google Maps Distance Matrix API est très similaire à Google Maps Destinations API pour ce qui est de la requête, sauf qu'on peut indiquer plusieurs points de départs et plusieurs points d'arrivée. La différence se trouve surtout dans la réponse qui ne donne qu'une matrice contenant les distances et les durées de parcours pour chaque trajet, sans les itinéraires. Cela ne nous intéresse donc pas (Google, 2017b).

Google Maps JavaScript API permet de mettre toutes sortes d'éléments sur une carte. Elle permet aussi d'utiliser les deux services décrits ci-dessus, Directions et Distance Matrix. Elle n'apporte rien de plus mais est spécialement utile pour la programmation en JavaScript (Google, 2017d).

Aucun service de Google ne fait les calculs d'un trajet qui contournerait un obstacle ni ne nous permet d'obtenir le graphe du réseau routier par lequel nous pourrions effectuer nous-mêmes le calcul (Google, 2017d).

L'application développée par l'Institut que nous devons étendre utilise quant à elle les cartes de Google Maps.

1.5.2. ViaMichelin

ViaMichelin est une autre application en ligne permettant le calcul d'itinéraires pour les trajets en véhicule privé, en moto, en vélo ou à pied. Elle ne connaît pas les parcours en transport public, ni les déplacements pour personnes à mobilité réduite. La Figure 12 illustre le parcours de la place du Tunnel à la place Saint-François à Lausanne ; il emprunte les escaliers du Marché, infranchissables pour une personne à mobilité réduite.

Figure 12 – ViaMichelin : itinéraire de la place du Tunnel à la place Saint-François, à Lausanne.

Les variantes 1 et 3 empruntent les escaliers du Marché

Source : capture sur fr.viamichelin.ch.

Viamichelin dispose par ailleurs de plusieurs API de calcul d'itinéraire, payantes. Il ne nous a pas été possible de découvrir les tarifs sans contacter directement Viamichelin.

1.5.3. OpenStreetMap

OpenStreetMap (OSM) est une base de données géographique libre. Elle a été fondée en juillet 2004 par Steve Coast à l'University College London. Elle fonctionne sur le même modèle que Wikipedia et compte sur la contribution des utilisateurs. Grâce aux outils de Global Positioning System (GPS), il n'y a plus besoin d'avoir une formation universitaire adéquate pour mesurer la terre et transcrire les informations sur le papier ou dans un ordinateur (Haklay & Weber, 2008, pp. 12-13). Steve Coast a remarqué que les données cartographiques du Royaume-Uni étaient très chères à acquérir auprès de l'*Ordnance Survey* qui, quoique financé par l'état, met toute une série de restrictions de licence sur ses cartes. C'est ce qui l'a incité à se lancer dans ce projet considérable (Brown, 2007). Il existe des associations nationales OSM pour soutenir le mouvement OSM dans leur pays. Par exemple, la Swiss OpenStreetMap Association (SOSM) a pour but :

> d'appuyer et de promouvoir des projets, des personnes, des entreprises et des organisations dans l'ensemble des régions linguistiques du pays, lesquels rassemblent, utilisent, élaborent et propagent des géodonnées libres et ouvertes. Ces activités peuvent être aussi bien de nature non-commerciales que commerciales. (Swiss OpenStreetMap Association [SOSM], 2012)

Les données d'OpenStreetMap sont stockées dans une base de données PostgreSQL depuis l'API v0.6, un système relationnel de gestion de base de données open source (OpenStreetMap, 2017e).

Il existe plusieurs API qui permettent de récupérer, et d'ailleurs aussi de modifier, les données dans la BD. Il y a en particulier une *overpass API*, en lecture seule, qui permet de récupérer une partie sélectionnée de données. OSM dispose d'un wiki avec les informations nécessaires pour son utilisation (OpenStreetMap, 2017b) (OpenStreetMap, 2017d).

A noter que OpenStreetMap connaît la notion de *barrier*. On peut en particulier définir une marche de trottoir *kerb*, avec sa hauteur *heigth*=* (OpenStreetMap, 2017a).

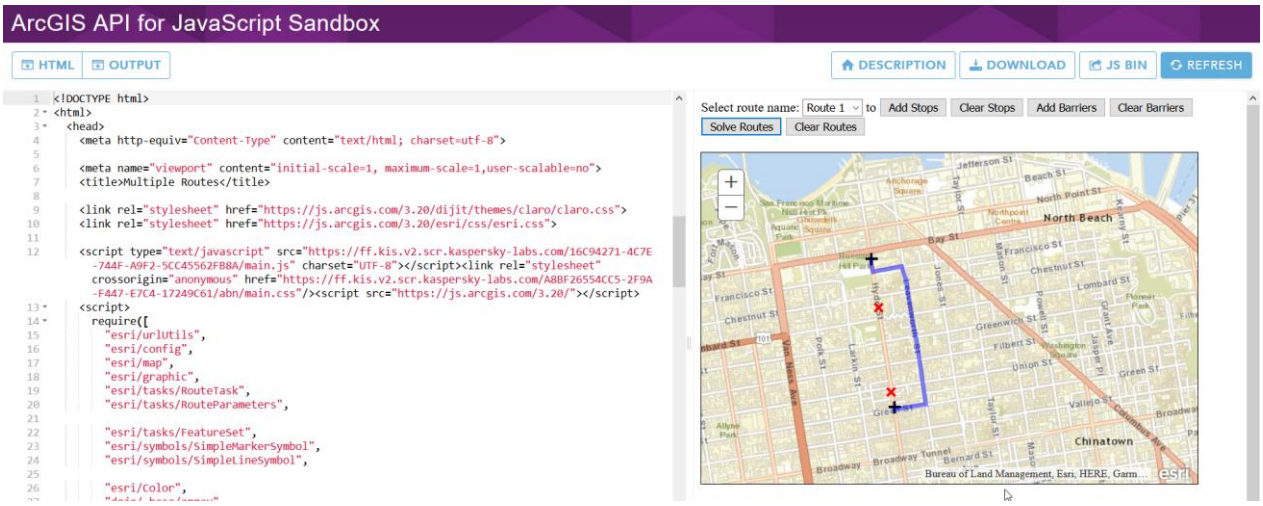
Le défaut de cette solution est évidemment la question de la fiabilité et de l'état lacunaire des sources. L'avantage principal est qu'elle est d'accès parfaitement gratuit. Beaucoup d'applications dont nous allons parler plus tard (chapitre 1.6) utilisent ses données cartographiques.

1.5.4. ArcGIS

ArcGIS est une suite de logiciels formant un système d'information géographique (SIG), en anglais un Geographic information system (GIS), développé par la société ESRI (pour Environmental Systems Research Institute, Inc.), basée à Redlands, en Californie. Ce système est payant. Il ne nous a pas été possible de connaître le coût exact, car il faut non seulement ouvrir un compte, mais même souscrire un essai gratuit de 60 jours pour pouvoir ensuite s'abonner pour une année (ESRI, s.d.).

Il est clair que ce logiciel permettrait de faire ce que nous souhaitons faire, comme le montre son bac à sable tel qu'on le voit dans la Figure 13, qui illustre comment il est possible de mettre un point de départ, un point d'arrivée et des obstacles (*barriers*) ; le chemin calculé, en bleu, part d'une des croix noires, évite les deux obstacles représentés par les croix rouges et se termine à la seconde croix noire.

Figure 13 – ArcGIS : bac à sable permettant de tester l'application en ligne.



Source : capture sur
https://developers.arcgis.com/javascript/3/sandbox/sandbox.html?sample=routetask_multiple_stops

1.5.5. Office fédéral de topographie Swisstopo

Mises à part les cartes consultables en ligne, l'Office fédéral de topographie Swisstopo offre en OpenData 65 jeux de données. Parmi ceux-ci, il y a des cartes numérisées, mais malheureusement uniquement aux formats 1:1 million et 1:500'000. Ces formats ne nous sont d'aucune utilité car seuls les grands axes routiers sont indiqués (Opendata swiss, s.d.).

Swisstopo met aussi à disposition des API qui permettent d'afficher des cartes dans un site Internet. On peut ajouter ensuite des objets sur la carte. Il n'y a pas de calcul d'itinéraires et encore moins de mise à disposition de données brutes (Swisstopo, s.d.).

1.5.6. Comparaison des différents systèmes de cartes

Les différents systèmes de cartes que nous venons de voir offrent différents avantages et inconvénients. Le Tableau 4 résume cette situation. Les systèmes propriétaires Google Map, Via Michelin, ArcGIS et les données de la Confédération offrent l'avantage de la fiabilité des données, au contraire d'OpenStreetMap qui est dépendant de ses contributeurs. À l'exception d'ArcGIS et de Viamichelin pour ses API, tous les systèmes sont gratuits, au moins jusqu'à un certain point. Le calcul d'itinéraires existe pour chaque système, à l'exception des données de l'Office fédéral de la topographie, mais rien n'est prévu pour les personnes à mobilité réduite. Il est en général possible de choisir le mode de transport et c'est la marche à pied qui se rapproche le plus de ce que nous souhaitons. ArcGIS gère parfaitement le contournement d'obstacle. Avec OpenStreetMap, tout est possible, le système étant ouvert et un certain nombre de librairies open source étant disponibles.

Tableau 4 – Caractéristiques des différents systèmes de cartes

Application	Open source	Fiable	Gratuit	Données brutes disponibles	Calcul d'itinéraire		
					Dispo-nible	Mobilité réduite	Obstacles
Google Map	✗	✓	(✓)	✗	✓	✗	✗
Via Michelin	✗	✓	✓✗	✗	✓	✗	✗
OpenStreetMap	✓	✗	✓	✓	✓	✗	(✓)
ArcGIS	✗	✓	✗	✗	✓	✗	✓
Swisstopo OpenData	✓	✓	✓	✓	✗	✗	✗
Swisstopo API	✗	✓	?	✗	✗	✗	✗

Source : tableau de l'auteur.

Il ressort de ces considérations que deux possibilités s'offrent à nous. Soit nous voulons vraiment pouvoir calculer des déplacements en évitant absolument des obstacles. Dans ce cas, la seule possibilité est de travailler sur OpenStreetMap, avec des librairies qu'il reste à définir à ce niveau. Soit nous demandons à un outil comme Google Maps de calculer plusieurs propositions d'itinéraires et nous analysons ensuite les résultats sous l'angle de la mobilité réduite. Mais si aucune des propositions de Google Maps n'est envisageable pour l'utilisateur final, il ne sera pas possible de lui calculer un itinéraire de remplacement.

Il est aussi peut-être envisageable de combiner les deux solutions.

1.6. Librairies de traitement de cartes

Sur la page Framework du Wiki d'OpenStreetMap, section Navigation, une série de librairies permettant de faire du calcul d'itinéraires est donnée. Le Tableau 5 en présente quelques-unes (OpenStreetMap, 2017c).

C'est seulement en testant ces librairies qu'il serait possible de se rendre compte de ce qu'elles pourraient vraiment apporter à notre projet.

Tableau 5 – Librairies spécialisées dans le calcul d'itinéraires dans OpenStreetMap.

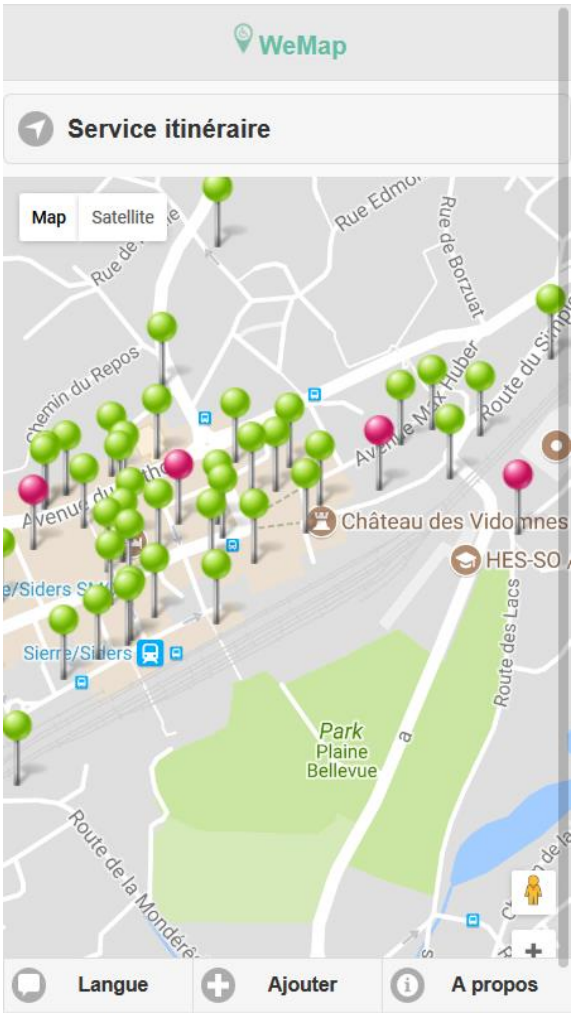
Librairie	Plateformes	Langages	License	Algorithme	Notes
CartoType	Windows, iOS, Android, macOS, Unix	C++, .NET, Java, Objective-C++	Propriétaire	Contraction hiérarchique[a]	
GraphHopper	Multiplateforme	Java, Objective-C, Swift	Apache 2	Dijkstra, A*, contraction hiérarchique[b]	
Mapbox GL Directions	Web	JavaScript	MIT	Inconnu	Plugin pour Mapbox GL JS
Libosmscout	Multiplateforme	C++, Java	LGPL	A* (?)[c]	Permet de définir des coûts sur mesure pour des segments de route[d]
pgRouting[e]	Multiplateforme	C++		Dijkstra, A* etc…	Basé sur PostGIS

Source : tableau de l'auteur d'après OpenStreetMap (2017c) et CartoType (2016)[a], karussell (2017) [b], Teulings, Mailing Lists (2012)[c], Teulings, Features. Routing (2016)[d] et pgRouting Community (s.d.)[e].

2. Développement

Au début de notre travail, l'application développée à l'Institut d'informatique de gestion de la HES-SO Valais-Wallis permettait déjà de référencer des points (POI ou routes) dans la base de données. Elle affichait ensuite ces points sur une carte. En vert figuraient les points n'ayant que des caractéristiques positives (accès sans marche, parking ou toilettes pour handicapés) et en rouge les points ayant au moins une des caractéristiques négatives (pente supérieure à 10°, trottoir de moins d'un mètre et demi et obstacles). Quelques centaines de points avaient été saisis par les facteurs dans les régions de Sierre, Sion et du Val d'Anniviers. La Figure 14 présente un exemple de cette carte. L'application est programmée en HTML5, PHP, JavaScript et MySQL. Elle est donc disponible sur Internet, utilisable sur ordinateur ou sur smartphone au moyen d'un navigateur Internet.

Figure 14 – Carte de l'application WE-MAP existante, dans la région entre la gare de Sierre et la HES-SO Valais-Wallis à Sierre.



Source : capture d'écran de l'auteur.

Quand l'utilisateur clique sur l'un des marqueurs, rouge ou vert, une fenêtre s'ouvre qui donne des informations plus détaillées sur le point concerné. Nous verrons cela plus en détail au point 2.5.

L'application que nous avons reçue utilisait Google Maps pour référencer les POI et les routes avec leurs caractéristiques respectives. Aucune solution pleinement satisfaisante n'ayant été trouvée dans la première partie de ce travail pour le calcul d'itinéraires, nous avons donc fait le choix de la cohérence en faisant une première approche avec ce même outil.

2.1. Google Maps

2.1.1. Google Maps Directions API

Tableau 6 – Paramètres de Google Maps Directions API.

*Les paramètres précédés d'une *étoile sont obligatoires. Les valeurs possibles suivies d'un dièse# sont les valeurs par défaut. Il y a d'autres paramètres qui ne concernent pas les itinéraires à pied que nous n'avons pas retranscrits dans ce tableau.*

Paramètre	Description	Valeurs possibles ou exemples
***origin**	Adresse, latitude/longitude ou identifiant de lieu de départ de l'itinéraire	*Exemples :* Rue+de+la+plaine+2+Sierre 46.597243,6.772480 place_id:ChIJ3S-JXmauEms RUclaWtf4MzE
***destination**	*Idem* pour le lieu d'arrivée	*Idem*
***key**	Clef d'API de l'application	
mode	Moyen de transport	• driving# • walking • bicycling • transit (=transports publics)
waypoints	Tableau de points de cheminement	
alternatives	Si ce paramètre est à « true », Directions peut proposer plusieurs itinéraires	• false# • true
avoid	La route calculée doit éviter certaines caractéristiques	• tolls • highways • ferries • indoor
language	Langue dans laquelle est retournée la réponse	
units	Système d'unités utilisé	• metric • imperial (par défaut système du pays ou de la région de départ)
region	Permet de limiter la recherche à une région	

Source : Tableau de l'auteur d'après Google (2017a).

La première étape consiste à récupérer les routes produites par Google Maps. Il existe deux outils de calcul de routes. Le premier est Google Maps Directions API. Cette API *REpresentational State Transfer* (REST) dont l'URL est https://maps.googleapis.com/maps/api/directions/json fonctionne avec un certain nombre de paramètres que l'on trouve listés dans le Tableau 6 . On peut aussi remplacer le dernier terme de l'URL, « json », par « xml » si l'on souhaite une réponse en eXtensible Markup Language (XML) plutôt qu'en JavaScript Object Notation (JSON). Google recommande cependant le JSON (Google, 2017a).

La saisie d'une requête dans un navigateur fonctionne très bien. La Figure 15 illustre le début du résultat de la requête https://maps.googleapis.com/maps/api/directions/json?origin=46.597243,6.772480&destination=46.594460,6.768069&mode=walking&alternatives=true. On peut noter que cette requête fonctionne sans le paramètre key, pourtant mentionné comme obligatoire dans la documentation.

Figure 15 – Google Maps Directions API : exemple de réponse.

Début de la réponse à la requête .


```
{
   "geocoded_waypoints" : [
      {
         "geocoder_status" : "OK",
         "place_id" : "EjNSdWUgZHUgR8OpbsOpcmFsIEd1aXNhbiAxMiwgMTA4MyBNw6l6acOocmVzLCBTdWlzc2U",
         "types" : [ "street_address" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJ42yjbwMrjEcRcO1sBBOlo20",
         "types" : [ "street_address" ]
      }
   ],
   "routes" : [
      {
         "bounds" : {
            "northeast" : {
               "lat" : 46.5975488,
               "lng" : 6.772052299999999
            },
            "southwest" : {
               "lat" : 46.5944645,
               "lng" : 6.768068100000001
            }
         },
         "copyrights" : "Données cartographiques ©2017 Google",
         "legs" : [
```


Source : capture d'écran de l'auteur, obtenue de Google Maps Javascript API.

L'application existante étant en HTML5/PHP/JS/MySQL, nous aurions souhaité pouvoir interroger cette API de Google en JavaScript, en Asynchronous JAvascript and XML (AJAX). Il s'agit d'un ensemble de technologies qui permettent de mettre à jour une page Web sans devoir la charger à nouveau (Delamarck & Pardanaud, 2017). AJAX utilise les langages que

nous venons de mentionner. Il utilise en particulier l'objet XMLHttpRequest qui, contrairement à ce que laisse entendre son nom, ne se limite pas du tout à des fichiers XML. Cet objet est inclus dans tous les navigateurs modernes. Il permet de préparer une requête http et de l'envoyer, puis d'écouter les événements qui se produisent durant le traitement de la requête et de déclencher une fonction quand ils surviennent (lessonsharing, et al., 2017).

Malheureusement, cette API applique la *same origin policy*. Cette politique, qui vise la sécurité, interdit les requêtes *cross-domain*, c'est-à-dire les requêtes qui n'ont pas la même origine. Par même origine, il faut entendre même protocole, même nom de domaine (ou de sous-domaine) et même port (Same Origin Policy, 2010).

Ainsi le code de la Figure 16 n'apporte-il aucune réponse à la requête. Google Maps Directions API est réservée à un usage côté serveur.

Figure 16 – Google Maps Directions API : exemple de code AJAX.

Ce code ne fonctionne pas en raison de la same origin policy *de l'API.*

```
<script>
    // HTTP request creation
    var req = new XMLHttpRequest();
    // HTTP GET request
    req.open("GET", "https://maps.googleapis.com/maps/api
                        /directions/json?origin=46.597243,6.772480
                        &destination=46.594460,6.768069
                        &mode=walking&alternatives=true", false);
    // End request event management
    req.addEventListener("load", function () {
        // Displays the answer for the request in the console
        console.log(req.responseText);
    });
    req.send(null);
</script>
```

Source : code de l'auteur.

2.1.2. Google Maps Javascript API – Service Directions

Nous devons donc nous tourner vers une autre application de Google : Google Maps Javascript API. Cette API permet la personnalisation en Javascript des cartes Google Maps. Elle comporte en particulier un service Directions, comparable à la Google Maps Directions API. Pour des raisons de gestion de quota, il faut une clef d'API Google pour pouvoir utiliser cette API. Google autorise 2'500 requêtes quotidiennes gratuites, pouvant comporter jusqu'à 23 points de cheminement (*waypoints*) chacune. Le nombre de requêtes par seconde est limité à 50. Moyennant USD 0.50 par 1'000 requêtes, il est possible d'augmenter le quota journalier jusqu'à 100'000. Il est enfin possible d'augmenter ces plafonds en optant pour un plan Premium, à des conditions que nous n'avons pas étudiées (Google, 2017e).

Pour utiliser cette API, il faut créer un objet de type *DirectionsService*. On lui applique ensuite la méthode *route()*, qui prend deux arguments. Le premier est un objet littéral *DirectionsRequest*. Le Tableau 7 en décrit les champs avec leurs caractéristiques. Le second argument est la méthode de rappel qui doit s'exécuter à la fin de la requête. En effet, l'appel au serveur externe rend la requête asynchrone ; il faut donc attendre d'avoir reçu la réponse pour pouvoir l'exploiter, mais sans pour autant bloquer le programme (Google, 2017e).

Tableau 7 – Champs de l'objet littéral DirectionsRequest.

*Les champs précédés d'une *étoile sont obligatoires. Il y a d'autres champs qui ne concernent pas les itinéraires à pied.*

Champ	Type	Notes
***origin**	LatLng String google.maps.Place	Point de départ
***destination**	LatLng String google.maps.Place	Point d'arrivée
***travelMode**	TravelMode	Peut être 'DRIVING', 'BICYCLING', 'TRANSIT' ou 'WALKING'. Nous choisirons 'WALKING'
unitSystem	UnitSystem	UnitSystem.METRIC ou UnitSystem.IMPERIAL. Par défaut, utilise le système du pays du point de départ.
waypoints[ ]	DirectionsWaypoint	Le waypoint a deux champs, *location* qui peut être des mêmes types que *origin* et *stopover*.
optimizeWaypoints	Boolean	Si ce champ est à *true*, le service va optimiser l'ordre des waypoints. C'est une application du problème du voyageur de commerce.
provideRouteAlternatives	Boolean	Si ce champ est à *true*, le service peut retourner plusieurs itinéraires dans un tableau.
avoidHighways	Boolean	Permet d'éviter les grands axes routiers.

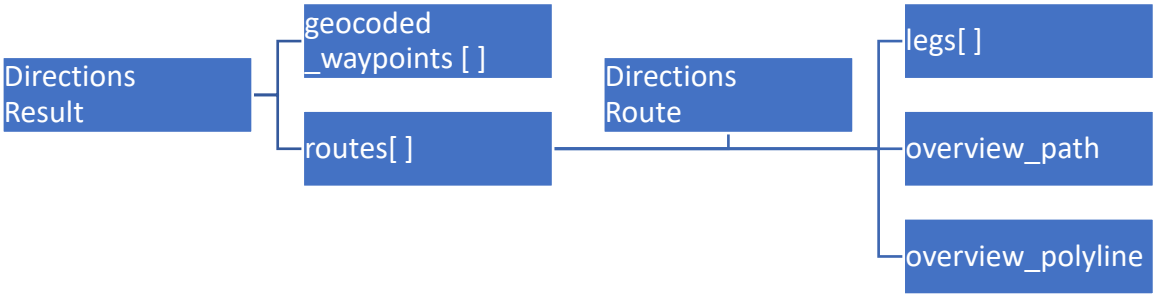
Source : tableau de l'auteur d'après Google (2017e).

La fonction de rappel qui forme le deuxième argument de la méthode *route()* a elle-même deux arguments : le résultat, un *DirectionsResult* et le status, un *DirectionsStatus*. Si ce dernier est égal à 'OK', c'est que le résultat est valide. Les autres valeurs possibles sont 'NOT_FOUND', 'ZERO_RESULTS', 'MAX_WAYPOINTS_EXCEEDED', 'INVALID_REQUEST', 'OVER_QUERY_LIMIT', 'REQUEST_DENIED' et 'UNKNOWN_ERROR' (Google, 2017e).

L'objet littéral *DirectionsResult* qui forme le résultat de la requête, contient un tableau de points géocodés, *geocoded_waypoints[ ]*, qui contient le point de départ, le point d'arrivée et les points de cheminement, chacun sous la forme d'un objet *DirectionsGeocodedWaypoint* et

routes[ ], un tableau d'itinéraires sous la forme d'objet littéral *DirectionsRoute*. L'Annexe II illustre la complexité de l'objet littéral *DirectionsResult*. Une explication détaillée des différents champs et objets est donnée par Google (2017e) sur son site pour développeurs. La Figure 17 en montre les éléments principaux. L'*overview_polyline* contient un seul objet *points* qui représente l'itinéraire sous forme de polyligne encodée, alors que l'*overview_path* contient un tableau d'objets *LatLng*. Ces deux objets représentent un tracé lissé de l'itinéraire concerné. C'est le second, l'*overview_path* que nous allons utiliser par la suite.

Figure 17 – Principaux éléments de l'objet littéral DirectionsResult.



Source : illustration de l'auteur d'après Google (2017e).

Si l'on veut afficher l'itinéraire sur la carte, il est possible de créer un objet *DirectionsRenderer*, d'attribuer à son champ *directions* le *DirectionResult*, à son champ *map* la carte sur laquelle afficher et si on veut afficher une route en particulier, il faut choisir le *routeIndex*. Par défaut, c'est la première route (*routeIndex*: 0) qui s'affiche. Un exemple de *DirectionsRenderer* se trouve en Figure 18.

Figure 18 – Exemple de DirectionsRenderer.

```
//display the itinerary
new google.maps.DirectionsRenderer({
    map: map,
    directions: response,
    routeIndex: 1,
    polylineOptions: {strokeColor: colors[1], clickable: false}

})
```

Source : code de l'auteur, d'après Google (2017c).

Il faut noter que chaque route retournée contient systématiquement un tableau *warnings*, dont le premier et unique élément est le message : « Walking directions are in beta. Use caution – This route may be missing sidewalks or pedestrian paths. ». Google ne prend donc aucune responsabilité quant à la qualité des trajets piétonniers.

2.1.3. Google Maps JavaScript API – *LatLng* et *Polyline*

Nous venons de voir que l'origine, la destination et les points de cheminement pouvaient être de type *LatLng*. Comme nous avons utilisé cet objet par la suite, il vaut la peine de donner quelques précisions le concernant. Il est défini dans la classe google.maps.LatLng. Il représente un point définit par ses coordonnées géographiques de latitude et longitude. La latitude est comprise dans l'intervalle [–90,+90] et la longitude dans l'intervalle [–180,+180]. Une fois un LatLng instancié, on ne peut pas modifier les valeurs de longitude et de latitude. Si nécessaire, il faut créer un nouvel objet LatLng. Le constructeur admet trois paramètres. Les deux premiers sont des nombres, la latitude et la longitude, toujours dans cet ordre. Le troisième, booléen, est optionnel et est à *false* par défaut. Si on le met à *true*, l'objet acceptera des latitudes et longitudes en dehors des intervalles prévus. Sinon, ces valeurs seront converties selon des règles qu'il ne vaut pas la peine de mentionner ici, mais que le lecteur retrouvera dans les informations de Google (2017g), section #LatLng. Il existe un objet littéral LatLngLiteral, qui peut être passé au constructeur à la place des paramètres. Il est aussi accepté par la plupart des méthodes qui acceptent LatLng. Il comporte deux champs, *lat* et *lng*.

Un autre objet de google.maps que nous utiliserons est la *Polyline*. Cette polyligne représente une série de segments connectés sur la carte. Elle se construit avec un seul paramètre, un objet *PolylineOptions*, dont les propriétés seront passées à la polyligne. Parmi ces propriétés, nous pouvons citer *clickable*, un booléen par défaut *true* qui indique si la *polyline* gère les événements de souris ; *map*, la carte sur laquelle afficher la polyligne ; *path*, un tableau de LatLng (ou de LatLngLiteral) ; *strokeColor*, la couleur d'affichage du trait (Google, 2017g).

2.1.4. Google Maps JavaScript API – Bibliothèque *Geometry*

Dans le cadre de notre projet, nous souhaitons déterminer si un point de notre base de données, POI ou route, est situé sur l'un ou l'autre trajet calculé par Google. Pour cela, nous disposons de la bibiliothèque *Geometry* de Google Maps JavaScript API. Cette bibliothèque a trois *namespaces*, spherical, encoding et poly. C'est ce dernier qui nous intéresse, car il permet de faire des calculs sur les poylignes et les polygones, et plus particulièrement sa méthode *isLocationOnEdge()*. Sa signature est la suivante :

```
isLocationOnEdge(point:LatLng, poly:Polygon|Polyline, tolerance?:number).
```

On voit qu'elle accepte trois paramètres. Le premier est le point dont on cherche à déterminer s'il est sur le trajet ou non. Il s'agit d'un objet LatLng. Le second est la polyligne que nous voulons examiner. On voit que cela pourrait aussi être un polygone si l'on veut savoir si un

point est compris dans un polygone. Dans notre cas, cependant, c'est une polyligne correspondant à chaque itinéraire calculé par Directions de Google Maps Javascript API. Le dernier paramètre, optionnel, représente la tolérance exprimée en degré. La valeur par défaut de la tolérance est de 10^{-9} degrés. La méthode retourne *true* si la distance entre le point choisi et le point le plus proche de la polyligne sont éloignés d'une distance plus petite que la tolérance (Google, 2017f).

Pour construire l'objet LatLng indiquant le point, il s'agit de récupérer les latitude et longitude des emplacements stockés dans notre base de données et d'utiliser le constructeur de cet objet. Cela ne présente pas de difficulté particulière. Par exemple, si *lat* et *lng* sont les nombres représentant ces coordonnées géographiques, l'instanciation aura la forme suivante :

```
let point = new google.maps.LatLng(lat, lng);
```

La polyligne est obtenue à partir d'une des routes retournées par le service *Directions*. Il s'agit d'un objet *Polyline* au constructeur duquel il faut transmettre l'*overview_path* de chaque route. Le paramètre du constructeur est un *PolylineOptions*, un objet qui a de nombreuses propriétés, dont *path*, qui peut être un tableau de *LatLng*, par exemple *overview_path*. On construit donc la polyligne de la façon suivante :

```
let polyline = new google.maps.Polyline({
                    path: response.routes[i].overview_path
});
```

La tolérance est exprimée en degrés et la valeur par défaut est de 10^{-9} degrés, selon Google (2017f). Nous en déduisons que si le point (46.281486,7.510887) est sur la polyligne et que la tolérance est de 0.000001, le point (46.281488,7.510887) n'y sera plus (pour autant que la ligne soit d'est en ouest). Il serait utile d'avoir une correspondance entre les degrés et les distances métriques. Cela n'est malheureusement pas simple. Si l'on considère que la circonférence de la Terre est d'environ 40'000 km et qu'un cercle fait 360° cela donne $\frac{40'000km}{360°} = 111.11\,\mathrm{km}$ par degré. Dans ce cas, la valeur par défaut de 10^{-9} degrés correspondrait à 0.111 mm. Cette valeur est une bonne estimation pour les degrés de latitude, qui se mesurent le long des méridiens, tous de longueur égale. Pour la longitude cependant, qui se mesure le long des parallèles, la longueur d'un degré n'est pas constante. Elle est d'environ 111 km à l'équateur, mais plus on monte en latitude, plus elle se raccourcit. Au pôle, elle est nulle. Considérons par exemple le 46e parallèle nord, le plus proche de Sierre : son rayon *r* se calcule ainsi : $r = \cos 46° \cdot r_T$, si r_T est le rayon de la Terre. La longueur du 46e parallèle ($2\pi r$) est donc égale à celle de l'équateur ($2\pi r_T$) multipliée par $\cos 46°$ et un degré mesure $111.1\,\mathrm{km} \cdot \cos 46° = 77{,}2\,\mathrm{km}$. Pour le parallèle suivant, le résultat est de

$111.1\ \mathrm{km} \cdot \cos 47° = 75.8\ \mathrm{km}$; on constate qu'un degré fait déjà 1.4 km de moins. Il ressort de ces calculs que si la plus courte distance entre un point et une polyligne se mesure sur un axe nord-sud, la tolérance est exprimée en multiple de 111 km. Si elle se mesure sur un axe est-ouest, sous nos latitudes, il s'agira d'un multiple de 76.5±0.7 km. Il ne nous a pas été possible de découvrir comment Google effectue les calculs pour les distances en diagonale. Nous nous baserons sur un ordre de grandeur de 10^{-3} pour un hectomètre, 10^{-5} pour un mètre.

2.2. Sélection des points à analyser

Comme à terme notre base de données va s'enrichir de très nombreux points, il ne sert à rien de les comparer tous aux routes calculées. Si, par exemple, l'utilisateur demande à l'application de calculer le trajet du Technopole à la gare, à Sierre, nous pressentons qu'il n'est pas utile de vérifier si les emplacements enregistrés à Sion ou Vissoie sont sur le parcours ou à proximité. Cette intuition ne nous dit cependant pas où est la limite. Nous devons donc résoudre deux problèmes : quels points doivent être analysés et comment les sélectionner.

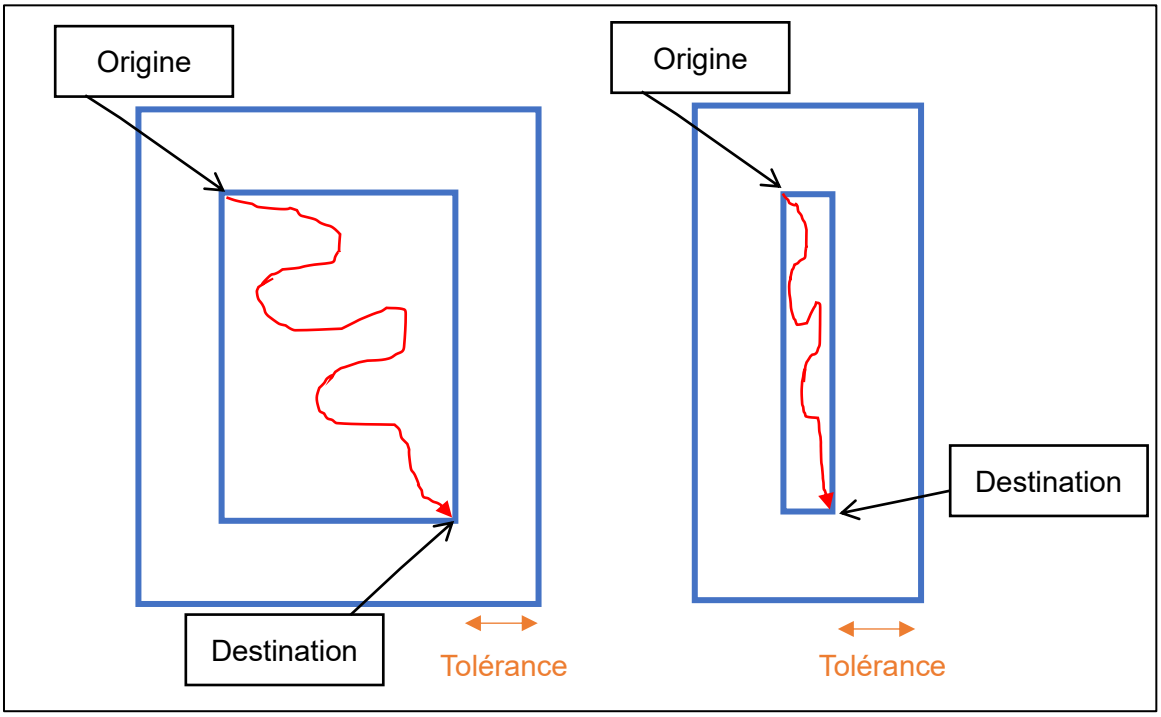
2.2.1. Choix des points à extraire dans la base de données

Les points sont stockés dans la base de données avec leurs coordonnées de longitude et de latitude exprimées en degrés. Nous avons imaginé plusieurs options pour opérer une présélection des points.

Nous aurions pu garder tous les points qui se trouvaient dans le rectangle dont l'origine et la destination du trajet forment deux angles opposés et dont les côtés vont d'est en ouest et du nord au sud. C'est d'ailleurs un faux rectangle, car la surface de la Terre n'est pas un plan. Mais vu que les trajets qui seront parcourus par les utilisateurs de notre application ne dépasseront *a priori* pas quelques kilomètres, nous pouvons accepter cette approximation. Il faudrait agrandir ce rectangle afin de tenir compte de la tolérance que nous donnerons à la méthode isLocationOnEdge(). Mais cela ne suffira pas, car le chemin peut lui-même sortir du rectangle. Il faudrait donc prévoir un rectangle encore plus grand. Ce rectangle offre l'avantage qu'il est facile de sélectionner les points. Il faut garder tous les points dont les latitudes sont comprises entre deux bornes et dont les longitudes sont comprises entre deux autres bornes. Mais l'inconvénient est que le rectangle sera presque carré si l'écart de latitudes et de longitudes entre l'origine et la destination sont proches et qu'il sera très étroit si l'origine et la destination ont des latitudes ou des longitudes proches, comme l'illustre la Figure 19. Dans le premier cas, les points se situant dans les deux autres angles du rectangle seront conservés inutilement. Dans le second cas, il y a de fortes chances que le trajet lui-même sorte du rectangle. En conclusion ce rectangle aura des proportions aléatoires en fonction de la position relative des origine et destination.

Figure 19 – Sélection des points dans la base de données : tracé d'un rectangle ayant l'origine et la destination comme angles opposés.

À gauche, les deux angles ont des longitudes et latitudes éloignées. À droite, les longitudes sont relativement proches en comparaison avec les latitudes.

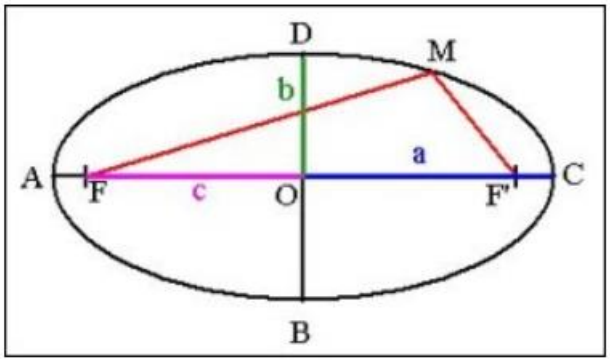


Source : illustration de l'auteur.

Une autre option serait d'avoir un rectangle dont deux des côtés sont parallèles à l'axe reliant l'origine à la destination. Il faudrait définir quelles seraient les dimensions du rectangle. L'avantage sur le rectangle précédent serait que la forme du rectangle n'est plus fonction des écarts relatifs de latitude et longitude. L'inconvénient est que le calcul de l'appartenance ou non au rectangle pour chaque point est obtenu par une équation plus compliquée.

Une troisième approche serait d'extraire tous les points se trouvant à l'intérieur d'une ellipse dont les foyers seraient l'origine et la destination. Dans ce cas, il faudrait définir le grand axe de l'ellipse (AC sur la Figure 20) comme, au minimum, la distance entre l'origine et la destination additionnée de deux fois la tolérance. Une ellipse peut être définie comme l'ensemble des points du plan dont la somme des distances à chacun des foyers est égale au grand axe. Sur la Figure 20, c'est l'ensemble des points *M* qui répondent à l'équation : $d(M,F) + d(M,F') = \overline{AC}$. Nous retiendrions donc tous les points dont la somme des distances aux deux foyers serait égale ou inférieure à la longueur du grand axe de l'ellipse.

Figure 20 – Sélection des points dans la base de données : ellipse.



Source : Ministère [français] de l'Éducation nationale (s.d.).

L'avantage de l'ellipse, c'est qu'elle va s'orienter en fonction de la position relative de l'origine et de la destination. Si la zone couverte ne semble pas suffisante, il suffit d'allonger le grand axe. Il faut cependant faire deux remarques. Premièrement, comme pour le rectangle, la surface de la Terre n'est pas un plan. Mais comme pour le rectangle, nous pouvons accepter cette approximation car les distances parcourues resteront faibles. En second lieu, nous l'avons vu, les coordonnées des points traités dans l'application, que ce soit l'origine et la destination ou les POI stockés dans notre base de données, sont toutes exprimées en degrés de longitude et de latitude. Il en va de même pour la tolérance. Or comme nous l'avons expliqué (2.1.4), un degré de latitude vaut 111,11 km sur toute la surface de la terre mais un degré de longitude varie en fonction de la latitude et vaut environ 76.5±0.7 km dans notre région. Si tous les calculs sont effectués en degrés ou en fractions de degrés, il y aura une déformation de l'ellipse, puisque les distances nord-sud seront plus longues que les distances est-ouest.

Tableau 8 – Extraction des points : principaux avantages et inconvénients de différentes méthodes de sélection.

	Avantages	Inconvénients
Rectangle orienté nord-sud et est-ouest	Calcul facile	Forme très variable en fonction des différences de latitude et longitude
		Les points près des deux autres angles sont gardés inutilement
Rectangle orienté selon l'axe origine-destination	La forme du rectangle ne dépend pas des différences relatives de latitude et longitude	Calcul compliqué
Ellipse dont les foyers sont l'origine et la destination	Englobe de manière cohérente la zone concernée	Si les calculs sont faits en degrés, les distances nord-sud sont plus longues que les distances est-ouest

Source : tableau de l'auteur.

Le Tableau 8 résume les avantages et inconvénients des trois méthodes de sélection. Notre choix s'est porté sur l'ellipse, ou plutôt la pseudo-ellipse, à cause de sa couverture cohérente de la zone concernée. Dans une ellipse, on nomme distance focale la distance qui sépare les deux foyers. Il s'agit donc ici de la distance entre l'origine et la destination. Nous nous sommes basés pour commencer sur un grand axe de deux fois la distance focale. Nous avons cependant vérifié dans le code qu'il ne soit pas inférieur à cette distance focale additionnée de deux fois la tolérance. En effet, si les deux points sont très proches en comparaison avec la tolérance, l'ellipse dont le grand axe serait la double distance focale serait trop petite pour englober tous les points qui sont dans la tolérance.

2.2.2. Sélection des points à afficher

L'application déjà existante affichait sur la carte l'ensemble des points. Elle utilisait pour cela l'objet google.maps.Marker de Google Maps JavaScript API. Il nous suffit donc de récupérer tous ces marqueurs dans un tableau *markersList*, puis d'analyser pour chacun de ces marqueurs s'il est dans l'ellipse définie au point précédent (2.2.1).

Dans un premier temps, pour le calcul des distances, nous avons créé une petite méthode dont le code figure en Figure 21. Le calcul utilise le théorème de Pythagore :

$$d(latitudes)^2 + d(longitudes)^2 = distance^2$$

$$\Rightarrow distance = \sqrt{d(latitudes)^2 + d(longitudes)^2}.$$

On calcule les différences de longitude et de latitude, puis on les met au carré et on prend la racine carrée de la somme. Cette méthode reçoit deux objets LatLng et retourne la distance exprimée en degrés.

Figure 21 – Méthode de calcul des distances.

```
/**
 * This function returns the distance in degrees between two points.
 * @param latlng1 (google.maps.LatLng)
 * @param latlng2 (google.maps.LatLng)
 * Returns the distance expressed as degrees. Using Pythagoras, take the
 * square-root of the sum of the square of the difference
 * of latitude and the square of the difference of longitude.
 */
function calculateDistance(latlng1, latlng2) {
    return Math.sqrt(Math.pow(latlng1.lat()-latlng2.lat(), 2) +
                              Math.pow(latlng1.lng()-latlng2.lng(), 2));
}
```

Source : code de l'auteur (route_service.js).

Il existe aussi dans la bibliothèque *Geometry* de Google Maps Javascript API, plus précisément dans l'espace de nom « spherical », une méthode *computeDistanceBetween*().

Cette méthode reçoit deux paramètres, les deux *LatLng* dont on veut connaître la distance qui les sépare et retourne cette distance, en mètres. Cette méthode offre l'avantage non négligeable de tenir compte de la courbure de la Terre et, surtout, elle rend les distances en mètres, ce qui évite les déformations dues à la différence entre les degrés de latitude et de longitude. Le seul inconvénient est que la tolérance est exprimée en degrés. Elle n'est donc pas compatible. Or, nous calculons le grand-axe de l'ellipse en tenant compte de la tolérance. Il aurait fallu la diviser par une somme comprise entre 111'110 m et 76'500±700 m. Afin d'être sûr de ne pas laisser de côté un point, il aurait été plus prudent de la diviser par le nombre le plus petit, soit 75'800 m. Mais il nous est paru plus simple de garder toutes les distances en degrés et d'éviter des conversions.

2.3. Implémentation dans l'application existante

Dans l'application déjà développée, quand l'utilisateur arrivait sur la carte, tous les points référencés dans notre base de données étaient visibles. Ils étaient affichés en rouge s'ils comportaient une pente trop raide (> 10°), un trottoir trop étroit (< 1.5 m) ou un obstacle. Dans tous les autres cas, ils étaient affichés en vert. La Figure 14, page 28, illustre cette situation.

Nous avons reçu pour l'insérer dans l'application existante, la structure d'un formulaire pouvant se collapser et permettant le choix du mode de transport (fauteuil roulant, piéton, cycliste, automobile), le point de départ et le point d'arrivée, ainsi que le type d'obstacle à afficher. Il comportait aussi un bouton « OK » permettant de déclencher la recherche.

2.3.1. Adaptation du menu

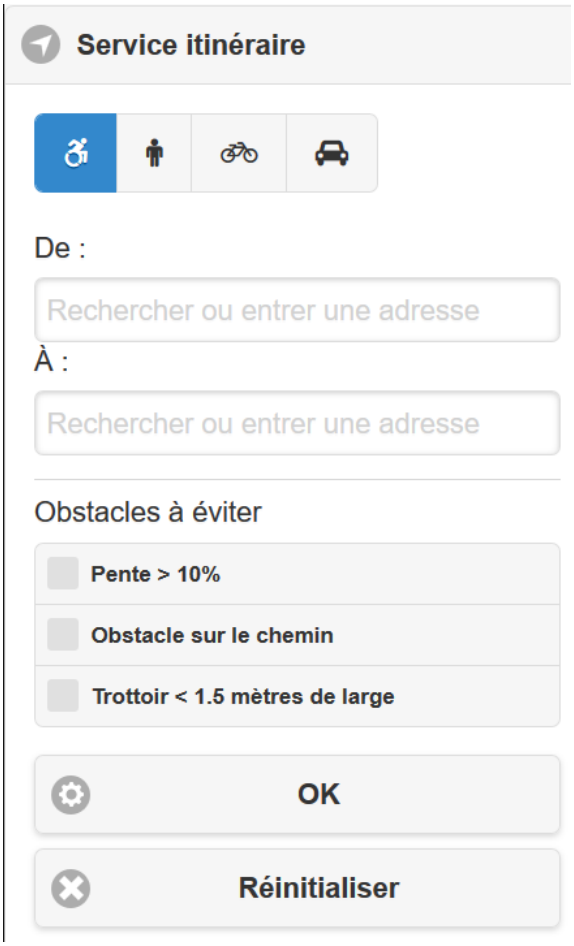
Il est prévu que l'utilisateur puisse choisir le mode de transport en cliquant sur une icône. Nous avons ajouté que l'option « fauteuil roulant » soit sélectionnée par défaut (checked) et nous avons associé aux quatre icônes les valeurs de « WHEELCHAIR », « WALKING », « BICYCLING » et « DRIVING ». Les trois dernières correspondent à des modes de transport prévus par Google. La première, nous l'avons ajoutée. Mais nous avons dû ensuite la convertir en « WALKING » lors du calcul d'itinéraires par Google pour que ce dernier puisse l'interpréter.

L'utilisateur sélectionne ensuite l'origine et la destination dans deux champs « De » et « À » qui s'auto-complèteront avec les données de Google.

L'utilisateur a ensuite la possibilité de cocher les types d'éléments qu'il souhaite voir s'afficher en rouge le long de sa route : pentes, obstacles, trottoirs trop étroits.

Ce formulaire est illustré par la Figure 22.

Figure 22 – Formulaire de saisie du Service itinéraire.



Source : illustration de l'auteur, capture d'écran.

2.3.2. Auto-complétion des champs « De » et « À »

Lorsque l'utilisateur clique sur le titre « Service itinéraire » afin d'ouvrir le menu de saisie, la fonction *initAutocomplete()* est lancée. Cette fonction utilise pour chacun des champs « De » et « À » un objet google.maps.places.Autocomplete, dont le constructeur prend en paramètre le champ qui doit bénéficier de l'autocomplétion. La méthode *bindTo*('bounds', map) que nous appliquons à l'objet *Autocomplete* est héritée de google.maps.MVCObject qu'étend *Autocomplete*. Elle permet de concentrer l'autocomplétion sur des points se trouvant sur la carte affichée, sans pour autant limiter la recherche à ce secteur. Cette méthode est l'objet de la Figure 23.

Figure 23 – Autocomplétion : la méthode initAutocomplete() *qui initialise l'autocomplétion des champs origine et destination.*

```
function initAutocomplete() {
    // Only first time...
    if(!init_autocompleted) {
        // Boolean testing if this method was already done
        init_autocompleted = true;

        // Get the origin (=from) field and put an autocomplete on it.
        let origin_field = document.getElementById("origin_field");
        autocomplete_origin =
                new google.maps.places.Autocomplete(origin_field);
        // This line directs the research (but doesn't limit it) to the map
        autocomplete_origin.bindTo('bounds', map);

        // Get the destination (=to) field and put an autocomplete on it.
        let destination_field =
                document.getElementById("destination_field");
        autocomplete_destination =
                new google.maps.places.Autocomplete(destination_field);
        // Directs the research to the map
        autocomplete_destination.bindTo('bounds', map);
      }
}
```

Source : code de l'auteur (route_service.js).

2.3.3. Méthode *searchRoutes()*

Cette méthode est le moteur principal de notre travail. C'est pour l'écrire que nous avons fait toutes les recherches qui ont précédé. Elle se divise en une première partie, qui est la récupération des données dans le formulaire rempli par l'utilisateur, et une seconde partie, qui est l'utilisation du *DirectionsService*. Cette deuxième partie consiste d'abord à récupérer et à afficher les itinéraires calculés, puis à traiter les marqueurs sur la carte.

2.3.3.1 Récupération des données

La première partie récupère les données dans le formulaire. D'abord elle prend le mode de transport, pour lequel il n'y a rien de particulier si ce n'est qu'il faut transformer « WHEELCHAIR » que Google ne connaît pas en « WALKING », le mode qui s'en rapproche le plus. Puis elle recueille les origine et destination, qui sont des objets LatLng que l'on obtient grâce aux objets Autocomplete avec, par exemple pour l'origine, l'instruction :

```
let origin = autocomplete_origin.getPlace().geometry.location;
```

Et enfin elle récupère les cases à cocher (*checkboxes*) indiquant le type d'éléments à afficher le long du chemin. Pour ce dernier point, nous avons stocké dans un objet littéral *checked_path_type* quels éléments ont été cochés. La Figure 24 montre l'instanciation de cet objet littéral. Les identifiants portent les mêmes noms que les id des cases à cocher correspondantes.

Figure 24 – Objet littéral checked_path_type *contenant les types d'éléments à afficher qui peuvent avoir été cochés.*

```
let checked_path_type = {
    slope: false,
    sidewalk: false,
    obstacle: false,
};
```

Source : code de l'auteur (route_service.js).

Nous avons ajouté dans le code qui nous a été fourni (map.js), dans les markers lors de leur création, en propriété, un objet semblable *path_types*, sujet de la Figure 25.

Figure 25 – Objet path_types *contenant les types d'obstacles et services potentiels.*

```
var path_types = {
    slope: false,
    sidewalk: false,
    obstacle: false,
    parking: false,
    entranceWithoutSteps: false,
    toilet: false
};
```

Source : code de l'auteur (map.js).

Nous pourrons ainsi aisément comparer les propriétés correspondantes des deux objets *path_types* – lié à chaque marqueur – et *checked_path_types* – lié aux choix de l'utilisateur. Si par la suite les responsables du projet souhaitent permettre à l'utilisateur de sélectionner aussi l'un ou l'autre des trois autres paramètres, à savoir la présence de place de parc ou de toilettes pour handicapés et une entrée sans marches, ou alors si un nouveau type de critère devait un jour être ajouté à l'application, il n'y aurait qu'à les ajouter à l'objet *checked_path_types*, avec les mêmes identifiants que dans *path_types*.

2.3.3.2 Utilisation du service *Directions*

La deuxième partie de la méthode *searchRoutes()* consiste à créer un DirectionsService et à l'alimenter avec une DirectionsRequest qui reprend les éléments issus de la première partie. Si la requête donne un résultat (status === 'OK'), elle affiche les différents chemins calculés. Nous avons cependant renoncé à utiliser un DirectionsRenderer, comme cité au point 2.1.2 et illustré par la Figure 18, page 33. Nous avons plutôt utilisé des polylignes stockées dans un tableau, en bouclant sur les itinéraires figurant dans le tableau *routes* de la réponse. Cela nous a permis de les manipuler ensuite plus aisément. Cette boucle apparaît dans la Figure 26. On y voit qu'en bouclant sur les itinéraires, on crée à chaque fois une polyligne qu'on ajoute au tableau *polylines[ ]*. On attribue à cette polyligne la carte *map* afin qu'elle y apparaisse. On lui attribue ensuite une couleur et on la rend *clickable*, c'est-à-dire qu'on autorise l'utilisateur à

cliquer dessus pour la sélectionner. Nous faisons la boucle en commençant par la dernière route, afin que la première apparaisse au premier plan, superposée aux autres.

Figure 26 – Boucle procédant à l'affichage des itinéraires au moyen de polylines.

```
// loop on each itinerary to get one polyline for each
for (let i = response.routes.length - 1; i >= 0 ; i--){

    // get the polyline and put it in an array
    polylines[i] = new google.maps.Polyline({
                                path: response.routes[i].overview_path});
    polylines[i].setMap(map);
    polylines[i].setOptions({strokeColor: colors[i], clickable: true});
    // ...
}
```

Source : code de l'auteur (route_service.js).

2.3.3.3 Traitement des marqueurs

Puis s'opère une boucle sur les marqueurs présents sur la carte. Pour cela, nous avons ajouté dans le code qui nous a été fourni (map.js) un tableau qui stocke chaque marqueur lors de sa création :

```
markersList.push(marker);
```

Avec cette solution, nous avons pu manipuler les marqueurs. En effet, il n'existe aucune méthode de Google Maps JavaScript API qui permette de récupérer les objets Markers sur une carte. Sans cela, nous aurions pu interroger à nouveau la base de données pour ajouter nos propres marqueurs, mais il nous aurait été impossible de faire disparaître les marqueurs déjà présents. Nous aurions dû ajouter une nouvelle carte ce qui n'aurait pas été rationnel. De plus, nous aurions dû récupérer une seconde fois les données sur la base de données, ce qui aurait manqué d'efficacité.

Un booléen *keepMarker* est mis à *false*. On commence par vérifier si le marqueur se trouve dans la pseudo-ellipse, au moyen de la fonction *isMarkerInEllipse*, dont le code se trouve en Figure 27. Nous rappelons que nous avons fait les choix suivants pour cette ellipse : ses foyers sont l'origine et la destination du trajet et son grand axe est égal à deux fois la distance qui les sépare, nommée distance focale. Nous contrôlons cependant que le grand axe de l'ellipse soit au moins égal à la somme de la distance focale additionnée de deux fois la tolérance. En effet, si par exemple la distance entre les deux points est de 100 m et que la tolérance est de 150 m, avec un grand axe de deux fois la distance focale, soit 200 m, on aurait des points à l'intérieur de la zone de tolérance qui se retrouveraient hors de l'ellipse.

Figure 27 – Fonction isMarkerInEllipse *qui détermine si un marqueur est dans l'ellipse dont les foyers sont l'origine et la destination.*

```
/**
 * This function calculate an ellipse, whom focuses are origin and
destination
 * @param marker (google.maps.Marker) : marker to control if it is in the
ellipse
 * @param origin (google.maps.LatLng) : origin of the calculated route
 * @param destination (google.maps.LatLng) : destination of the calculated
route
 * @param tolerance
 */
function isMarkerInEllipse(marker, origin, destination, tolerance) {
    /* define the ellipse */
    // distance between origin and destination (Pythagoras;
                // focal distance = distance to the center of ellipse)
    let doubleFocalDistance = calculateDistance(origin, destination);
    let majorAxis = 2 * doubleFocalDistance;
    if (majorAxis < doubleFocalDistance + 2 * tolerance)
        majorAxis = doubleFocalDistance + 2 * tolerance;

    // distance from marker to origin and destination (Pythagoras)
    let markerToOrigin = calculateDistance(origin, marker.getPosition());
    let markerToDestination = calculateDistance(destination,
marker.getPosition());

    // if marker is in the ellipse, the sum of the distances to each focus
                // is less than majorAxis and we return true
    return markerToOrigin + markerToDestination < majorAxis;
}
```

Source : code de l'auteur (route_service.js).

Si c'est le cas, on boucle sur les routes pour déterminer, au moyen de la méthode google.maps.geometry.poly.isLocationOnEdge(), si le marqueur est sur le chemin ou s'il en est proche, en tenant compte de la tolérance. Cet emboîtement de boucles et condition est l'objet de la Figure 28.

Si le marqueur ne remplit pas les conditions, il sera caché grâce à la ligne :

```
marker.setMap(null);
```

Le marqueur ainsi traité disparaît de la carte, mais il existe toujours dans le tableau *markersList*. Il pourra donc être réactivé facilement.

Figure 28 – Sélection des marqueurs qui ne doivent pas disparaître de la carte.

```
// Select points to display and hide other markers - Then choose color
                                                    // of kept markers.
// loop on all markers in the map
markersList.forEach(function (marker) {
    // boolean to control if the marker is on one of the routes
    let keepMarker = false;

    // Look if the marker is not far from the origin and destination.
        // If it is far, we don't have to waste a google request
    if (isMarkerInEllipse(marker, origin, destination, tolerance)) {
        // loop on routes
        for (let i = 0; i < response.routes.length; i++) {

            // This Google maps method look if the marker is on the
                        // polyline, according to the tolerance
            if (google.maps.geometry.poly.isLocationOnEdge(marker.position,
                        polylines[i], tolerance)) {
                // We keep the marker because it is on the itinerary
                keepMarker = true;
            }
        }
    }
    // hide the marker if it is not on one route
    if (!keepMarker){
        marker.setMap(null);
    } else {
        // ...
    }
})
```

Source : code de l'auteur (route_service.js).

Si le marqueur est conservé parce qu'il est proche de l'itinéraire, il faut décider s'il doit s'afficher en vert ou en rouge. En effet, par défaut, sur la carte, s'affichent en rouge tous les marqueurs qui représentent un endroit comportant soit une pente de plus de dix degrés, soit un trottoir trop étroit, soit un obstacle. Mais l'utilisateur, dans le calcul d'itinéraire, a le choix de tolérer l'une ou l'autre de ces contrariétés. L'élément concerné doit alors s'afficher en vert. Le code permettant de choisir la couleur est en Figure 29. Nous stockons dans une nouvelle propriété du marqueur l'icône d'origine. Il sera ainsi plus aisé de l'afficher à nouveau lors de la réinitialisation de la carte.

Figure 29 – Choix de la couleur des marqueurs

```
// Select points to display and hide other markers - Then choose color of
                                                        // kept markers.
// loop on all markers in the map
markersList.forEach(function (marker) {

    // ...

    if (!keepMarker){
        marker.setMap(null);
    } else {

        // Default color is green
        let markerColor = 'green';
        // Loop on the potentially checked path types (slope, sidewalk and
                // obstacle)
        for (let path_type in checked_path_types) {

            // Keep the first color of the marker
            if(!marker.hasOwnProperty('originIcon')) {
                marker.originIcon = marker.getIcon();
            }

            if (checked_path_types[path_type] &&
                        marker.path_types[path_type]) {

                    markerColor = 'red';
            }

            marker.setIcon('images/path_' + markerColor + '.png');
        }
    }
})
```

Source : code de l'auteur (route_service.js).

2.3.3.4 Recentrage de la carte

Il reste un détail : la carte doit se recentrer afin d'afficher les trajets de manière complètement visible. Pour cela, nous utilisons un objet google.maps.LatLngBounds. Cet objet représente un rectangle qui englobe tous les LatLng qu'il contient. Nous le déclarons immédiatement après avoir constaté que la requête avait obtenu une réponse valide :

```
var bounds = new google.maps.LatLngBounds();
```

Ensuite, dès que les polylignes sont créées, nous ajoutons tous les points qu'elles traversent au LatLngBounds :

```
polylines[i].getPath().forEach(function (latlng) {
   bounds.extend(latlng);
});
```

Et à la fin de la méthode, nous adaptons l'affichage de la carte pour qu'elle montre l'entier des itinéraires :

```
map.fitBounds(bounds);
```

2.3.3.5 Justification de la présélection par la pseudo-ellipse

Nous nous sommes demandé s'il était vraiment utile de présélectionner les points dans une pseudo-ellipse avant de calculer s'ils sont sur la polyligne. En effet, ce traitement nécessite de passer chaque point dans la méthode *isMarkerInEllipse()*, méthode qui doit à chaque fois définir l'ellipse et déterminer si le point est à l'intérieur. La question se pose s'il ne serait pas plus efficace de comparer directement le point avec la polyligne, plutôt que de perdre du temps à éliminer une partie des points pour ensuite vérifier si chacun d'eux se trouve sur la polyligne.

Cependant, dans les tests que nous avons faits, quand nous avons demandé au service Directions de retourner plusieurs routes en mettant la propriété *provideRouteAlternative* à *true*, l'expérience nous a montré que ce sont la plupart du temps trois routes, parfois deux, qui sont retournées. Cela signifie que tous les points doivent être confrontés à trois routes. Il est donc vraisemblable qu'il vaille la peine d'opérer une présélection. Comme nous ne pouvons pas nous baser sur des vraisemblances, nous avons souhaité effectuer un test sur la fonction *searchRoutes()*. Mais cette méthode fait appel à un *DirectionsService* dont l'utilisation est limitée, nous le rappelons (cf. 2.1.2), à un quota de 2'500 appels par jour et 50 par seconde, nous ne pouvons donc pas faire plus de 50 appels à la fois. Nous avons donc décidé de faire un test très simple uniquement sur la boucle *markersList.forEach()*. Nous l'avons mise dans une boucle pour l'effectuer un certain nombre de fois et avons relevé l'heure au début et à la fin. Les modifications – provisoires – de code faites pour tester ce temps se trouvent en Figure 30.

Figure 30 – Modifications provisoires du code pour la mesure de la durée du traitement des marqueurs.

```
var start = new Date().getTime();
for (let y = 0; y < 500; y++) {

    // loop on all markers in the map
    markersList.forEach(function (marker) {
        // ...
}

var end = new Date().getTime();
console.log("timer :", end, "-", start, "=", end - start);
```

Source : code de l'auteur (route_service.js provisoire).

Nous avons effectué les tests avec plusieurs navigateurs, en faisant respectivement 1000 puis 500 boucles. Nous avons chaque fois fait le test d'abord avec utilisation de l'ellipse, puis sans, dans l'ordre du Tableau 9 qui contient les résultats. Pour tester sans l'ellipse, nous avons remplacé la ligne :

```
if (isMarkerInEllipse(marker, origin, destination, tolerance)) {
```

par la ligne :

```
if (true) {
```

sans faire d'autre modification, afin d'éviter de fausser le test. Ainsi, tous les marqueurs étaient confrontés à chaque polyligne.

À chaque fois, nous avons demandé le calcul du trajet qui va de la « HES SO Valais Wallis / Informatique de gestion - Wirtschaftsinformatik, Rue de la Plaine, Sierre, Switzerland » au « Techno-Pôle Sierre SA, Sierre, Switzerland » qui a retourné systématiquement trois routes.

Tableau 9 – Temps nécessaire à l'exécution de la boucle markersList.forEach() *plusieurs fois.*

Navigateur	Boucles	Avec présélection	Sans présélection	Rapport
Chrome	1'000	2'474 ms	13'269 ms	5.4
Chrome	500	1'185 ms	7'431 ms	6.3
Firefox	1'000	4'377 ms	25'489 ms	5.8
Firefox	500	1'943 ms	12'800 ms	6.6
Opera	1'000	2'349 ms	14'921 ms	6.4
Opera	500	1'199 ms	6'869 ms	5.7
Edge	1'000	17'255 ms	104'929 ms	6.1
Edge	500	8'930 ms	51'780 ms	5.8

Source : données récoltées par l'auteur.

On constate dans le Tableau 9, mis à part les différences d'efficacité des navigateurs, que l'utilisation de l'ellipse pour la présélection se justifie pleinement, puisqu'elle a diminué le temps de calcul d'un facteur allant de 5.4 à 6.6. Il est raisonnable de penser que si l'application se développe comme nous l'espèrons et que le nombre de points se multiplie dans toute la Suisse et au-delà, ce facteur sera encore plus grand. En effet, le nombre de points qui seront à proximité du trajet, *a priori* de quelques kilomètres au maximum, devrait rester du même ordre de grandeur alors qu'il y aura des milliers de points référencés. Le nombre de points proches du trajet sera donc encore bien plus faible par rapport au nombre de points qui en sont éloignés.

2.3.3.6 Sélection par l'utilisateur de l'un des chemins affichés

Nous souhaitions donner à l'utilisateur la possibilité de cliquer sur l'une ou l'autre des routes affichées, afin de la mettre en évidence et d'afficher une petite étiquette (une google.maps.InfoWindow) avec la longueur du trajet et sa durée.

La première étape a consisté à créer cette étiquette. Pour cela, nous avons ajouté dans la boucle qui procède à l'affichage des itinéraires le code de la Figure 31. Il faut noter que les routes de la réponse de l'API ne contiennent qu'un seul *leg* si la requête ne comporte qu'un

point de départ et un point d'arrivée sans étape intermédiaire. C'est la raison pour laquelle nous avons pu lier la distance et la durée du premier (et unique) *leg* à toute la route.

Une difficulté réside dans la durée du trajet, lorsque le moyen de transport choisi est la chaise roulante. En effet, Google nous retourne le temps nécessaire à un piéton, puisque nous avons remplacé « WHEELCHAIR », non reconnu par Google, par « WALKING ». La durée du trajet qui nous est fournie est donc valable pour un piéton.

Une solution idéale aurait été de connaître un facteur par lequel multiplier cette durée pour obtenir la vitesse du fauteuil roulant. Nous avons donc cherché à déterminer la vitesse moyenne d'un fauteuil roulant. Sonenblum, Sprigle, & Lopez parlent d'une vitesse médiane de 0.43 m/s (2012, p. 5) pour un fauteuil roulant manuel, soit 1.55 km/h. Tolerico et al. (2007, p. 567) ont mesuré une vitesse moyenne de 0.79±0.19 m/s pour des vétérans, ce qui correspond à 2.84 km/h. Nous constatons donc qu'il peut y avoir de grandes différences pour les seuls fauteuils manuels, sans compter les fauteuils électriques qui vont plus vite.

Nous ne savons pas comment Google calcule le temps pour un piéton, mais nous constatons que ce n'est pas seulement en fonction de la distance. Par exemple, pour le trajet de la HES, route de la Plaine, au Technopôle, à Sierre, Google nous propose trois itinéraires. Le premier emprunte la route de la Plaine, puis la route de Sous-Géronde ; il est donc relativement à plat ; pour 1.4 km il met 19 minutes, soit 13.6 minutes par km (4.41 km/h). Le deuxième emprunte la route des Lacs, puis rejoint la route Sous-Géronde ; il franchit donc une colline ; il met 20 minutes pour franchir 1.5 km, soit 13.3 minutes par km (4.51 km/h). Le troisième parcours emprunte lui aussi la route des Lacs, mais il bifurque par la rue de Planzette, dont la pente est plus forte, pour franchir une seconde colline ; il met lui 22 minutes pour 1.5 km, soit 14.7 minutes par km (4.17 km/h). D'autres facteurs que la pente, par exemple les carrefours, entrent peut-être aussi dans le calcul. Il n'est donc pas évident de multiplier la durée reçue par Google par un facteur pour obtenir la vitesse d'un fauteuil roulant.

En raison de ces considérations, il nous a semblé plus judicieux de ne pas indiquer de durée de trajet lorsque l'utilisateur a sélectionné le type de transport « WHEELCHAIR ».

Nous avons choisi d'afficher l'étiquette à la fin du *step* du milieu, afin qu'elle soit plus ou moins au milieu de la route. Il aurait aussi été possible de la lier au point où l'utilisateur a cliqué. Par contre la faire apparaître à mi-distance entre l'origine et la destination n'aurait pas été une option ; la route n'étant pas forcément rectiligne, l'étiquette aurait pu s'en trouver fort éloignée.

Figure 31 – Création de l'étiquette liée à l'itinéraire calculé.

```
// Set an InfoWindow to each poly-line (the poly-line has one leg if there
                                                    // is no 'via').
// Prepare label text with distance and duration / wheelchairs : no duration
let labelTxt = response.routes[i].legs[0].distance.text;
if (travelMode !== "WHEELCHAIR") {
    labelTxt += '<BR>';
    labelTxt += response.routes[i].legs[0].duration.text;
}

// Calculate the location of the label : end of the step of the middle.
let steps = response.routes[i].legs[0].steps;
let centralStep = steps[Math.floor(steps.length/2)];
// Create the label
labels[i] = new google.maps.InfoWindow({
    content: labelTxt,
    position: centralStep.end_location
})
```

Source : code de l'auteur (route_service.js).

Nous avons créé ensuite dans la même boucle un *listener* qui réagira au clic sur une des polylignes. En Figure 32 se trouve ce code. Le paramètre « zIndex » des options de polylignes définit le plan dans lequel apparaissent les objets. Plus le nombre est grand, plus l'objet est au premier plan. Nous avons donc réinitialisé à zéro les zIndex de toutes les polylignes, puis nous avons mis à 1 celui de celle qui a été cliquée. Dans le même mouvement, nous avons changé les couleurs pour griser les polylignes, sauf celle qui est cliquée qui reste en couleur. Nous avons ensuite fermé toutes les étiquettes qui auraient pu être ouvertes et avons ouvert celle qui correspond à la polyligne cliquée.

Figure 32 – Listener *permettant de traiter le clic sur un des trajets calculés.*

```
// This click-listener allows to put in foreground the clicked polyline.
google.maps.event.addListener(polylines[i], 'click', function (clickEvent) {
    // put all polylines in background
    for (let j = 0; j < polylines.length; j++) {
        polylines[j].setOptions({zIndex: 0});
    }
    // put this polyline in frontground
    this.setOptions({zIndex: 1});

    // Hide all open labels:
    labels.forEach(function (label) {
        label.close();
    })
    // Display the label corresponding to the clicked polyline (this).
    labels[polylines.indexOf(this)].open(map);
})
```

Source : code de l'auteur (route_service.js).

2.3.4. Fonction *reinitializeRoutes()*

Nous avons créé une fonction *reinitializeRoutes()* qui est appelée à deux occasions : quand l'utilisateur clique sur le bouton « Réinitialiser » ainsi qu'au début de la méthode *searchRoutes()*, si des routes ont déjà été calculées et affichées sur la carte et que des marqueurs ont disparu.

Cette fonction, dont le code est en Figure 33, a donc pour but d'effacer les itinéraires qui figurent déjà sur la carte et de faire réapparaître tous les marqueurs, dans leur couleur d'origine. Elle contient une boucle sur les polylignes pour les ôter de la carte au moyen de l'instruction : « polyline.setMap(null); ». Le tableau *polylines[ ]* est ensuite vidé : « polylines.length = 0; ». Une seconde boucle ferme les étiquettes au moyen de l'instruction : « label.close(); » et le tableau *labels[ ]* est à son tour vidé (« labels.length = 0; »). Une dernière boucle sur les marqueurs les fait réapparaître sur la carte (« marker.setMap(map); ») et leur redonne, le cas échéant, leur couleur d'origine qui avait été stockée dans la propriété *originIcon*, qui est elle-même supprimée.

Figure 33 – Fonction reinitializeRoutes().

```
/**
 * This function removes all routes from the map and displays all markers
 */
function reinitializeRoutes() {
    // Collapse the Route Service menu:
    $("#routeSerivce").collapsible("collapse");

    // Remove all polylines from the map
    polylines.forEach(function (polyline) {
        polyline.setMap(null);
    })
    // remove all polylines from the polylines array
    polylines.length = 0;
    // remove all labels from the map
    labels.forEach(function (label) {
        label.close();
    })
    // remove all labels from the labels array
    labels.length = 0;
    // show all markers
    markersList.forEach(function (marker){
        marker.setMap(map);
        if (marker.hasOwnProperty('originIcon')) {
            marker.setIcon(marker.originIcon);
            delete(marker.originIcon);
        }
    })
}
```

Source : code de l'auteur (route_service.js).

2.4. Ajout de marqueurs au début et à la fin de la route

Nous avons ajouté au début et à la fin des trajets deux marqueurs, marqués respectivement « A » et « B ». Ces marqueurs ne se posent pas directement aux points d'origine et de destination, mais bien aux deux extrémités de la route. En effet, cette dernière commence devant le bâtiment qui a été sélectionné, sur la route, et pas directement dans le bâtiment. C'est pourquoi, dans le code affiché dans la Figure 34, on ne met pas dans la position simplement « origin » et « destination », mais bien « response.routes[0].legs[0].start_location » et « response.routes[0].legs[0].end_location ». Nous pouvons choisir la première route, car toutes les routes ont le même lieu de début et de fin. Comme mentionné au point 2.3.3.6, les routes ne comportent qu'un *leg* pour autant que la requête n'ait comporté qu'un point de départ et un point d'arrivée, sans étape intermédiaire. C'est pourquoi nous pouvons prendre non seulement le point de départ du premier leg, mais aussi son point d'arrivée. Si un développement ultérieur de l'application devait permettre de mettre des étapes intermédiaires, il faudrait alors préciser qu'on prend la fin du dernier *leg*, « legs[legs.length-1].end_location ». Mais il est probable que, dans ce cas, on voudra aussi mettre un marqueur à chaque étape et que le code sera donc entièrement repris. Par souci de clarté et de simplification, nous avons gardé la première option.

Figure 34 – Insertion des marqueurs au début et à la fin des routes.

```
// Put markers at the start and origin of first route (all routes have 1
                // leg, and same start and end)

markerOrigin = new google.maps.Marker({
    map: map,
    position: response.routes[0].legs[0].start_location,
    label: 'A'
})

markerDestination = new google.maps.Marker({
    map: map,
    position: response.routes[0].legs[0].end_location,
    label: 'B'
})
```

Source : code de l'auteur (route_service.js).

L'ajout de ces marqueurs nous oblige à les ôter lors de la réinitialisation de la route, sinon ils resteront sur la carte. Nous ajoutons donc deux lignes à la méthode *reinitializeRoutes()*, comme le montre la Figure 35.

Figure 35 – Retrait des marqueurs de début et de fin de route lors de la réinitialisation de la carte.

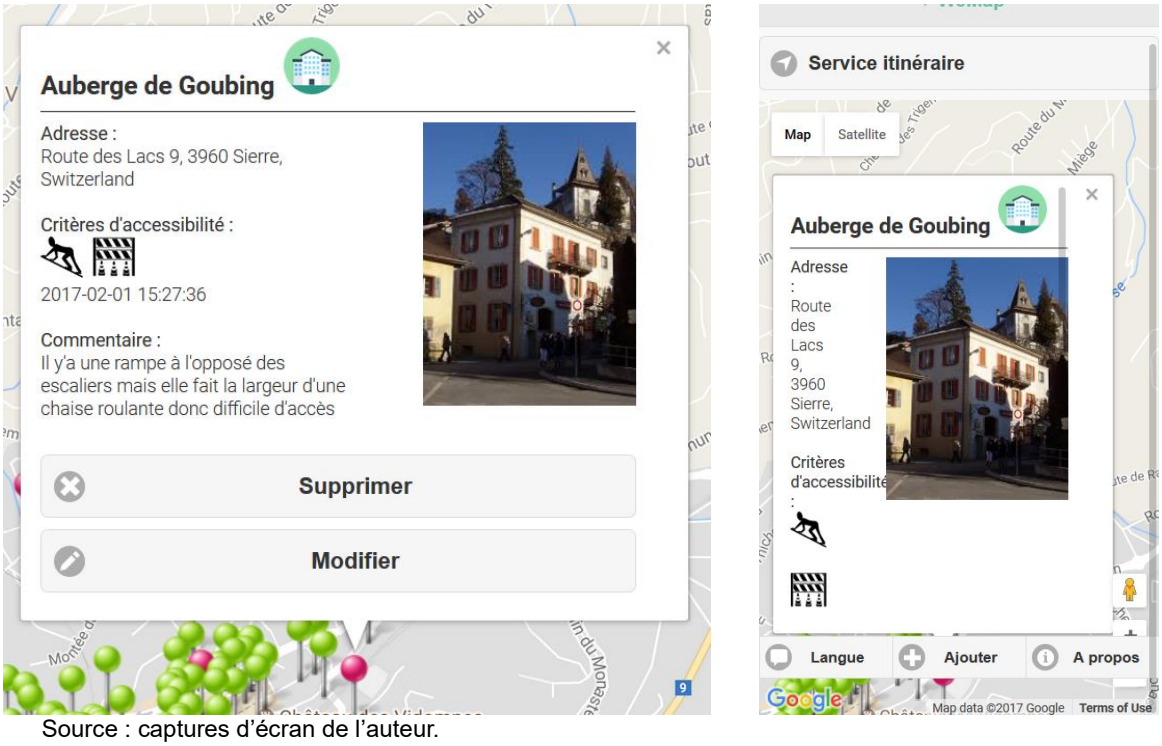
```
/**
 * This function reinitialise the map to the step before the routes
calculation
 */
function reinitializeRoutes() {
    // ...

    // remove the 'A' origin marker and 'B' destination marker from the map
    markerOrigin.setMap(null);
    markerDestination.setMap(null);

    // ...
}
```

Source : code de l'auteur (route_service.js).

2.5. *Modification des* InfoWindows *liées aux marqueurs*

Telle que l'application nous a été fournie, quand l'utilisateur cliquait sur un des marqueurs, que ce soit avant ou après le calcul de l'itinéraire, une fenêtre s'ouvrait avec des détails sur ce point. La Figure 36 montre cette fenêtre. À gauche, on voit la fenêtre telle qu'elle s'ouvre sur l'écran d'un ordinateur et à droite sur un téléphone portable.

Figure 36 – Description de l'Auberge de Goubing dans une InfoWindow*, à gauche sur un ordinateur personnel et à droite au format d'un écran de smartphone.*



Source : captures d'écran de l'auteur.

Nous avons relevé, avec les responsables du projet, des problèmes à deux niveaux, celui des critères d'accessibilité et celui de la mise en page. Tout d'abord, sur la fenêtre au format

smartphone, la place prise par la photo rend la colonne de texte à gauche trop étroite pour être pratique. Ensuite, pour les critères d'accessibilité, seuls les critères remplis (positifs ou négatifs) sont présents. De plus, les critères positifs (accès facile, place de parc, toilettes) et les critères négatifs (pente, trottoir trop étroit, obstacle) ne sont pas différenciés.

Cette InfoWindows est construite de manière dynamique dans le fichier map.js. Nous avons donc opéré dans ce fichier les modifications nécessaires pour que la photo apparaisse au sommet d'une colonne unique.

Nous avons ensuite modifié l'affichage des six icônes au moyen du code de la Figure 37. Ce code boucle sur les 6 critères pour savoir si l'icône doit apparaître en grisé ou en noir. Nous avons opté pour une opacité de 40 %.

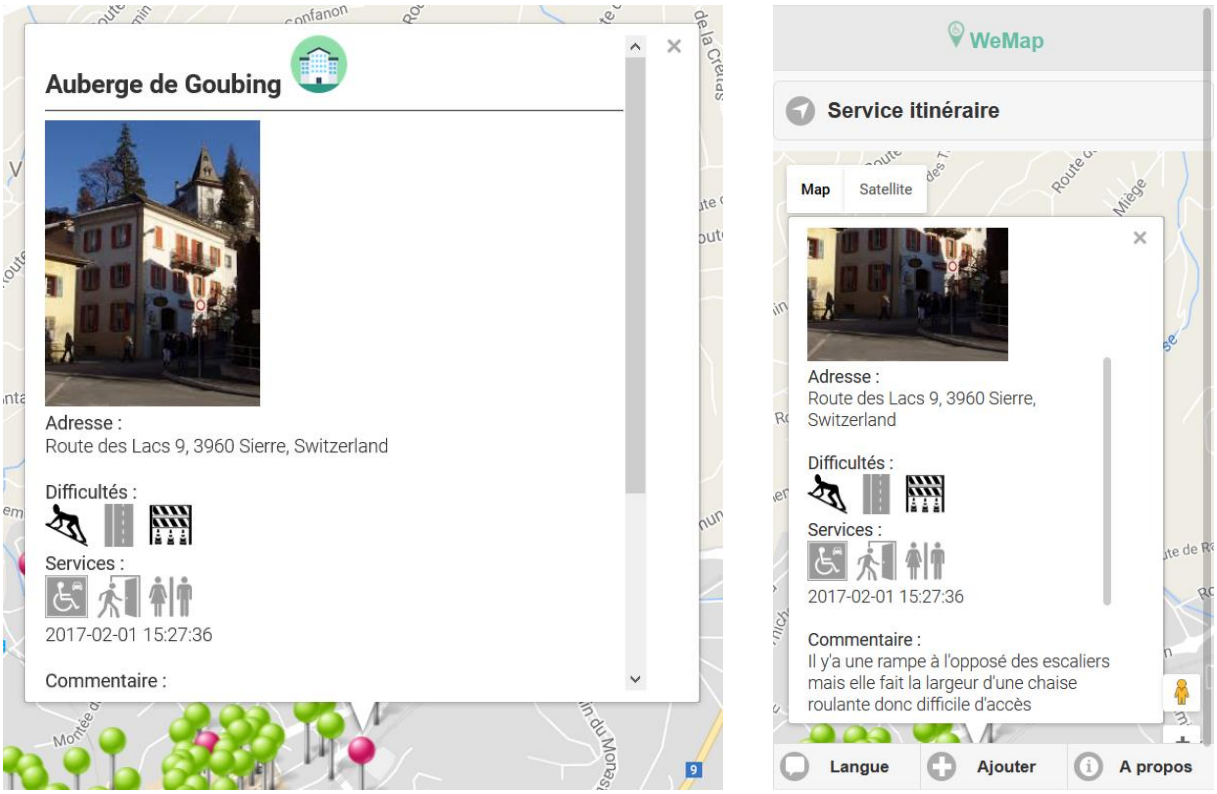
Figure 37 – Code d'affichage des six icônes.

```
for (let path_type in path_types){
    // Before services, write the services title (parking is the first
service)
    if(path_type === 'parking') {
        infotext += "<br><strong>" + lang['INFOBOX_SERVICES'] + " :
</strong><br>";
    }
    // ...
    // Put the image
    infotext += "<img src='images/" + path_type + ".png' alt='" + path_type
+ "' style='width:30px;height:30px;";
    // if the criterion is not present, the image will be transparent
    if(!path_types[path_type]) {
        infotext += "opacity: 0.4;filter: alpha(opacity=40);";
    }
    infotext += "'>";

    // ...
    infotext += "  ";
}
```

Source : code de l'auteur (map.js).

Le résultat de ces modifications apparaît dans la Figure 38. L'utilisateur est ainsi bien conscient des critères présents ou absents.

Figure 38 – Description de l'Auberge de Goubing après modification, à gauche sur un écran d'ordinateur, à droite au format d'un téléphone portable.



Source : captures de l'auteur.

2.6. Déploiement sur un serveur en ligne

Afin de rendre les tests possibles en déplacement, nous avons déployé l'application sur un serveur provisoire : www.famillemack.ch/wemap. Le déploiement s'est fait sans trop de difficultés. La base de données a pu être peuplée avec les données qui avaient été collectées par les facteurs de Sierre, Sion et du Val d'Anniviers grâce à un script SQL qui nous a été fourni. La seule difficulté rencontrée est que les *smartphones* avec iOS comme système d'exploitation n'acceptent pas de proposer à l'utilisateur sa géolocalisation si le site distant ne dispose pas d'un certificat SSL. Nous avons donc dû installer un tel certificat sur notre site, ce qui n'a pas posé de problème particulier, notre hébergeur nous proposant un certificat Let's Encrypt signé et gratuit. Nous avons ensuite ajouté le code de la Figure 39 au fichier .htaccess à la racine de notre serveur pour que l'utilisateur qui se connecte soit automatiquement redirigé sur le site sécurisé https (Infomaniak, s.d.).

Figure 39 – Code du fichier .htaccess *permettant de rediriger l'internaute vers le site sécurisé https.*

```
RewriteEngine on
RewriteCond %{HTTP:X-Forwarded-Proto} !https
RewriteRule (.*) https://www.famillemack.ch/$1 [R=301,L]
```

Source : code de l'auteur, d'après Infomaniak (s.d.)

3. Crowdsourcing

Notre application, pour l'instant, affiche des cartes et calcule des itinéraires. Ces fonctionnalités, en soi, ne sont pas très originales. On a vu (au chapitre 1.5) qu'il existe déjà bien des outils qui rendent ces services. La véritable valeur ajoutée de notre application, c'est la base de données avec les critères d'accessibilité et les services que nous pouvons mettre à disposition des utilisateurs. Pour qu'elle soit vraiment utile, il est indispensable qu'un très grand nombre de points d'intérêt et de routes y soient saisis. Il faut qu'elle soit la plus complète possible. Pour cela, nous comptons sur l'engagement des utilisateurs qui doivent saisir ces informations. C'est ce que l'on appelle le *crowdsourcing* : la « foule » des usagers va participer à la récolte d'information. Cela pose le problème de la motivation de ces usagers, qui doivent se mobiliser pour glaner un maximum de renseignements.

Cette partie du travail va donc porter sur les méthodes permettant d'inciter les utilisateurs à remplir notre base de données. À ce niveau du travail, nous n'irons pas jusqu'à l'implémentation, faute de temps. Nous allons nous concentrer sur la recherche de pistes qui pourront être utiles aux responsables du projet général.

3.1. Généralités sur le crowdsourcing

Le terme *crowdsourcing* est un mot-valise dérivé de *crowd*, la foule en anglais, et de *outsourcing*, l'externalisation ou sous-traitance. Il s'agit bien de sous-traiter un travail à la foule (Coline B, 2016). Ce néologisme a été créé par Jeff Howe qui l'a employé pour la première fois dans un article publié en juin 2006 dans *Wired*, un magazine de technologie américain (Sobczak & Groß, 2010, p. 15). À noter que la Commission générale de terminologie et néologie (2014, p. 12995) propose les termes de « production participative » ou « production collaborative » pour éviter l'anglicisme *crowdsourcing*.

Schenk et Guittard (2012, pp. 94-95) distinguent trois situations de *crowdsourcing* en fonction de ce qui est requis de la foule. Dans la première, on demande aux gens de réaliser une tâche simple, mais à grande échelle. Dans la deuxième, l'organisation cherche à résoudre un problème complexe ; il faut alors que les participants disposent de ressources cognitives spécifiques et rares ; ce doit donc être des experts. Dans la troisième, c'est la capacité créative des individus qui est sollicitée ; ce n'est alors ni le nombre, ni l'expertise des individus qui sont utiles. C'est bien le premier cas qui nous concerne. Nous souhaitons que les utilisateurs réalisent une tâche simple, qui ne demande pas de connaissances particulières, mais à de multiples reprises : la saisie des données pour chaque point.

Souvent, pour les tâches simples, la rémunération est nulle ou il s'agit de micro-paiements. Les problèmes complexes nécessitent eux une rémunération élevée de la foule. Quant aux

productions créatives, leur rémunération est très variable (Schenk & Guittard, 2012, pp. 95-97).

La motivation de la foule peut être intrinsèque, liée à la satisfaction d'accomplir la tâche pour elle-même ou pour son aspect social, ou extrinsèque, liée à une rémunération. À cela s'ajoute la contribution involontaire, dont nous parlerons au chapitre 3.2.1 (Schenk & Guittard, 2012, p. 97). La motivation extrinsèque peut aussi être liée à d'autres facteurs qu'une rémunération financière. Ce peut être la réputation, le prestige, la pression des pairs, la renommée, l'identification à une communauté, l'amusement (Hossain, 2012, pp. 502-504). Hossain (p. 504) a étudié 268 plateformes de *crowdsourcing* ; parmi elles 27.6 % avaient recours à une motivation intrinsèque, 49.6 % à une motivation extrinsèque financière et 22.8 % à une motivation extrinsèque non financière. Hossain (p. 503) n'est pas clair sur le classement de l'amusement : « In *crowdsourcing* platforms, the significant extrinsic motivational factors are […] and fun etc. Fun and enjoyment are considered as the two leading intrinsic motivational factors prevalent in online platforms ». Layas et Petrie (2016, p. 547) classent au contraire sans hésiter l'amusement dans les motivations intrinsèques, avec le challenge intellectuel, l'amour de la communauté ou le fait de passer le temps. Morschheuser, Hamari et Koivisto (2016, p. 3) le classent aussi dans les motivations intrinsèques.

Un aspect important cité par Schenk et Guittard (2012, pp. 97-98) est la confiance réciproque que doivent se faire les deux partenaires que sont l'entreprise ou la plateforme qui sollicite le crowdsourcing d'une part et l'utilisateur d'autre part. Il faut qu'il ait confiance dans l'utilisation qui sera faite de sa contribution. En particulier s'il n'est pas rémunéré il ne doit pas avoir l'impression que l'autre partie va exploiter de manière opportuniste sa participation. L'entreprise doit pouvoir avoir confiance dans les compétences et l'honnêteté des contributeurs.

3.2. Techniques de motivation au crowdsourcing

En préambule, en matière de motivation, il est difficile de trouver des techniques qui fonctionnent pour tous, car chaque individu est différent, a ses propres intérêts et est mu par des ressorts-clefs distincts qui se sont construits par son histoire personnelle, ses expériences, ses échecs (Segal & Duron, 2015, p. 254).

Comme il n'est pas question de rémunérer les contributeurs de WE-MAP, car il n'y a pas de budget pour cela, nous devons nous tourner vers la motivation extrinsèque non financière ou la motivation intrinsèque. Avant cela, nous allons citer la participation involontaire.

3.2.1. Participation involontaire

Schenk et Guittard (2012, pp. 95-97) citent l'exemple de la tâche simple de ReCaptcha. Un *Captcha* est un système qui permet de s'assurer que l'individu qui visite un site est bien un être humain et pas un robot. Il consiste à présenter une image comportant un mot déformé à l'utilisateur, en lui demandant de saisir ce mot. Une machine n'arrivera pas à reconnaître les caractères tordus. Un ReCaptcha comporte deux mots. Le premier est connu du système et sert à filtrer les robots. Mais le second est un mot qui a été rejeté par un logiciel de reconnaissance de caractères et qui est ainsi déchiffré par la foule des internautes. Google a racheté cette société et s'en sert pour son programme de numérisation d'ouvrages. La contribution de l'internaute qui pense simplement remplir un filtre anti-spam est donc involontaire et, bien sûr, n'est pas rémunérée. Une extension avec la reconnaissance de photos de plaques de rues a été mise en place depuis 2012 pour améliorer Google Street View (Champeau, 2012).

Cet exemple a frappé les esprits ; en effet, nous l'avons trouvé souvent cité (Andro & Saleh, 2015) (Michelucci & Dickinson, 2016) (Schenk & Guittard, 2012, p. 95). C'est un véritable succès. Selon son créateur, von Ahn (2011), 750 millions de personnes ont ainsi contribué à digitaliser les connaissances humaines, soit plus du dixième de l'humanité.

ReCaptcha cumule les avantages : c'est facile à utiliser, sa présence sur de nombreux sites lui permet d'atteindre une large audience, les qualités requises pour contribuer sont à la portée du plus grand nombre et la motivation marche à deux niveaux : l'utilisateur veut pouvoir accéder à un site et le webmaster veut écarter les spammeurs (Rosario, 2011).

Pour la plateforme qui utilise la participation involontaire pour le *crowdsourcing*, il n'y a que des avantages. Cela ne coûte rien. L'utilisateur ne se rendant même pas compte de sa contribution, il ne va pas répugner à la donner. Malheureusement, notre application ne nous permet pas d'utiliser ce type de *crowdsourcing*. Nous n'imaginons pas comment inciter l'utilisateur à saisir les informations utiles sans qu'il ne s'en rende compte. Les données demandées à l'utilisateur sont trop complexes pour cela.

3.2.2. Motivation intrinsèque

La motivation intrinsèque est liée à la satisfaction d'accomplir la tâche pour elle-même ou à son aspect social (Schenk & Guittard, 2012, p. 97). Dans le cas de WE-MAP, la motivation intrinsèque liée à l'aspect social est importante.

3.2.2.1 Aspect social

Un exemple très réussi de *crowdsourcing* à grande échelle est Wikipedia. Cela est vrai même si son créateur Jimmy Wales (2013) rejette ce terme dérivé de *outsourcing*, ce qui

implique selon lui l'idée de sous-traiter le travail pénible auprès d'une main d'œuvre bon marché. Cette encyclopédie en ligne utilise la foule des internautes qui contribue volontairement et par idéal.

Howe (2009, pp. 47-48), le créateur du terme *crowdsourcing*, cite l'exemple de Linux comme exemple précurseur de réussite de ce concept, au début des années 1990. Comme vraiment beaucoup de personnes ont contribué, la charge n'a pas été trop lourde comme elle l'aurait été pour quelques-uns.

Layas et Petrie (2016, pp. 553-554) ont étudié les motivations qui poussaient des gens à soutenir des étudiants mal-voyants au travers de DescribeIT, une application qui permet à des voyants de décrire à la demande des images de ressources pédagogiques numériques à des étudiants malvoyants. Elles ont montré que dans le groupe des personnes recevant un micro-paiement pour chaque image décrite (motivation extrinsèque financière) et dans le groupe d'étudiants dont les instructions insistaient sur l'amélioration de leurs propres compétences par cet exercice (motivation extrinsèque non financière), les participants étaient significativement motivés par le fait d'aider des étudiants handicapés (motivation intrinsèque). Cela surprend les auteures, car la motivation extrinsèque, en particulier financière, est souvent citée comme premier moteur du *crowdsourcing*. L'application WE-MAP est comparable à DescribeIT dans le sens qu'elle vise aussi à soutenir des personnes handicapées. Le fait d'aider les personnes à mobilité réduite peut être source de motivation.

WE-MAP diffère en revanche de DescribeIT par le fait que les personnes concernées peuvent elles-mêmes contribuer à la récolte de données. Quand elles fréquentent un bâtiment, l'existence appréciée de toilettes pour handicapés ou au contraire la présence malheureuse de marches rendant l'accès impossible vont immédiatement les frapper, car ces informations les touchent directement. Elles pourraient donc être motivées par l'aspect communautaire et souhaiter rendre service à d'autres personnes partageant leur situation, par solidarité. Nous n'avons cependant pas pu prouver à ce stade qu'une telle solidarité existe, c'est une hypothèse.

3.2.2.2 Ludification (*gamification*)

Le mot ludification, du latin *ludus*, qui désigne le jeu, est souvent remplacé en français par l'anglicisme *gamification*.

La ludification est un phénomène qui prend de l'ampleur en parallèle avec le *crowdsourcing*. Des analystes commerciaux ont estimé que la majorité des organisations ont « ludifié » certains de leurs processus en 2015. Ludification et *crowdsourcing* sont souvent liés, le second étant un des principaux secteurs d'application de la première. La ludification permet de donner

l'impression qu'on est plus en train de jouer que de travailler. Elle poursuit deux buts, augmenter la motivation intrinsèque à prendre part à l'activité et ensuite à augmenter cette activité (Morschheuser, Hamari, & Koivisto, 2016, pp. 1-2).

Morschheuser et al. (2016, pp. 3-7) ont fait une recherche de littérature sur la gamification. Ils ont sélectionné 28 articles. Lorsque, comme pour WE-MAP, on demande à la foule d'effectuer à de nombreuses reprises une tâche simple – les auteurs parlent de *crowdprocessing* – les systèmes basés sur la compétition avec des systèmes de points et de tableau de classement (*leaderboard*) sont très souvent utilisés. Viennent ensuite les systèmes de badges, de niveaux, de progression etc. La plupart des études analysées montrent que la ludification augmente l'engagement à long-terme, améliore la qualité de la production et réduit la tricherie par rapport au *crowdsourcing* rémunéré. Par contre, elle n'augmente pas nécessairement la participation, c'est-à-dire l'entrée dans le système. Les auteurs donnent quelques recommandations. Dans le *crowdprocessing*, les tâches sont faciles à compter et presque tous les systèmes analysés disposaient d'un système de points pour chaque tâche accomplie. Le tableau de classement peut être à double tranchant. Plusieurs études montrent qu'il faut faire des classements à court-terme, qui peuvent être très efficaces pour motiver certains participants. Au contraire, des classements à long-termes sont contre-productifs car ils peuvent décourager les participants placés au bas de l'échelle et les novices, pour qui le sommet du tableau semble inaccessible. Cependant, une étude relativise ce constat en se basant sur le fait que le *crowdsourcing* suit souvent le principe du « 90-9-1 » (ou règle du un pour cent), ce qui signifie que seul un pour cent des participants exécute la presque totalité des tâches ; dans ce cas, un tableau de classement à long-terme pourrait se justifier. Une des études citées par cet article semble donner un léger avantage en termes de motivation aux ludifications qui permettent de se comparer aux autres par rapport à celles qui fonctionnent avec des niveaux que l'on peut atteindre et qui illustrent uniquement la progression personnelle. Il semble enfin que multiplier les possibilités de jeu pour toucher plusieurs groupes cibles ne soit pas forcément utile dans le cas homogène du *crowdprocessing* et que les jeunes soient plus stimulés par les systèmes basés sur la compétition que leurs aînés. L'altruisme et la curiosité restent d'importants moteurs de la contribution. Les auteurs restent prudents dans leurs conclusions. Si la ludification améliore très vraisemblablement l'engagement et la qualité du résultat, le choix de l'implémentation qui va le mieux fonctionner pour une situation et des groupes de contributeurs donnés reste un défi.

En se basant sur ces informations, nous pourrions proposer pour notre application d'attribuer un point à chaque saisie de données. Les contributeurs pourraient ensuite afficher leur classement, par exemple, pour une localité, une région ou un canton. Ce classement devrait se faire sur le court terme, par exemple par mois, afin d'éviter de décourager les

nouveaux entrants ou ceux qui ont peu contribué au début. Dans un monde de réseaux sociaux, on pourrait aussi imaginer de pouvoir se comparer à ses amis et connaissances.

3.2.3. Motivation extrinsèque

Nous l'avons vu, la motivation extrinsèque peut être liée le plus souvent à une rémunération. Mais la réputation, le prestige, la pression des pairs, la renommée, l'identification à une communauté, l'amusement peuvent être d'autres sources de motivation pour la foule. Ce qui caractérise la motivation extrinsèque, c'est que l'individu qui participe au *crowdsourcing* reçoit en échange une récompense.

Il faut savoir que dans la suite du développement de l'application WE-MAP, les utilisateurs, qu'ils veuillent consulter les informations ou en étendre la matière, devront s'identifier. Il sera donc à l'avenir possible de savoir qui a contribué, de quelle manière et dans quelle mesure. Il sera donc aisé de récompenser chaque effort de la manière qui sera jugée la meilleure et la plus efficace pour garnir au mieux la base de données.

3.2.3.1 Récompense financière

Selon Hossain (2012, p. 504), la motivation financière est la plus répandue et concerne environ la moitié des plateformes de *crowdsourcing*. Comme la tâche que nous demandons à la foule est simple, mais qu'elle doit se répéter souvent, il ne pourrait s'agir que de micro-paiements. Par micro-paiement, nous entendons des paiements inférieurs à un dollar, souvent de quelques centimes (Schenk & Guittard, 2012, pp. 95-97).

Selon Grier, les personnes qui font du *crowdsourcing* souhaitent gagner un salaire horaire de l'ordre de 5 à 12 USD (2013, p. 131). La saisie d'une donnée prend moins d'une minute. Il serait théoriquement possible de rémunérer chaque contribution à hauteur de dix centimes, puisqu'on en connaîtra l'auteur. Les montants seraient accumulés et quand ils atteindraient une certaine somme, ils pourraient être versés au contributeur. Le paiement pourrait d'ailleurs être effectué en bons d'achats plutôt qu'en espèces, ce qui offrirait le double avantage d'être plus simple que de multiples transactions et d'intéresser des entreprises sponsors. En effet, les coûts totaux pour la plateforme pourraient rapidement devenir très importants et il faudrait bien trouver une manière de les financer. Au contraire, pour la foule, l'effort à accomplir pour réunir une somme intéressante peut paraître important, s'il faut saisir un millier d'information pour gagner 50 ou 100 francs.

Des variantes à ce modèle sont possibles. Plutôt que d'attribuer directement une valeur pécuniaire à chaque insertion d'une donnée, il serait envisageable d'utiliser un système de points. Le participant serait ainsi moins conscient du peu de valeur financière de chaque contribution. Quand il aurait accumulé un certain nombre de points, il pourrait choisir entre

plusieurs bons offerts par des sponsors. Il serait intéressant de faire une étude pour savoir quel serait le meilleur rapport entre la valeur des récompenses et le nombre de points nécessaires pour les obtenir, afin de motiver la foule au meilleur coût.

Une autre variante, plus économique, serait de tirer au sort les récompenses parmi les contributeurs les plus méritants. Mais il ne faudrait pas en arriver à décourager les participants parce qu'ils ne reçoivent jamais rien.

3.2.3.2 Récompense liée à l'usage de l'application elle-même

Il est aussi possible de limiter l'usage de l'application à quelques recherches d'itinéraires pour une période donnée, par exemple dix recherches par mois. Les personnes qui saisissent des données obtiendrait des points qui, au lieu de leur donner une récompense en espèces ou sous forme de bons, leur permettrait d'effectuer plus de recherches. La difficulté de l'exercice serait de déterminer d'une part quel devrait être le nombre de recherches de base autorisées et combien de saisies il faudrait effectuer pour avoir droit à une recherche supplémentaire. Pour que ce système soit incitatif, il faut que le nombre de recherches de base ne suffise pas à l'utilisateur, pour l'obliger à saisir des données. Mais il faut éviter que les recherches supplémentaires ne lui demandent un effort trop considérable. Sinon, il risque tout simplement de se détourner de l'application.

Ce système présente l'avantage qu'il est parfaitement gratuit pour la plateforme. Il a cependant plusieurs défauts. Il faut d'abord imaginer que lorsqu'un utilisateur très motivé aura saisi toutes les données potentielles dans sa région, il n'aura plus d'occasion de gagner des points et se verra privé du fruit de son travail, puisqu'il ne pourra plus profiter pleinement de l'application. Il faut aussi garder à l'esprit que certains utilisateurs à mobilité réduite seront simplement incapables d'alimenter la base de données, car ils ne peuvent justement pas se déplacer facilement pour visiter les régions encore inexplorées par l'application. Enfin, cette manière de faire ne motive que ceux qui ont besoin de l'application. Or, il serait préférable de pouvoir motiver le plus grand nombre de personnes, y compris les valides.

3.3. Limites de la motivation au crowdsourcing

Quelle que soit la méthode utilisée pour motiver les gens, il faut être conscient du risque d'une baisse de la qualité des informations au profit d'une augmentation du volume. Que ce soit pour gagner des places dans un classement en cas de ludification ou pour obtenir une meilleure rémunération en cas de motivation extrinsèque, des participants peuvent être tentés de saisir de nombreuses informations peu fiables dans le seul but de gagner des points.

Lebraty et Lobre mettent en garde contre un des dangers du *crowdsourcing* : « le côté avide des individus ». Alors qu'on parle souvent de la sagesse des foules en relation avec le *crowdsourcing*, ils mettent en garde contre la « bêtise des foules » (2015, pp. 113-114).

Seules les personnes motivées par l'altruisme et l'aspect social sont moins susceptibles de tomber dans ces travers qui seraient justement contraires à ces buts.

Bien heureusement, et comme le montrent Morschheuser et al. (2016, p. 7), les motivations intrinsèques et extrinsèques sont souvent mêlées. Ainsi, les personnes motivées par les aspects ludiques ou pécuniers ne seront pas forcément dépourvues d'altruisme ou de curiosité. Cela devrait les inciter à ne pas privilégier seulement le volume de données, mais bien aussi la qualité.

Il vaut quand même mieux rester vigilant et disposer de méthodes de contrôle des informations, soit en recoupant les informations de plusieurs contributeurs, soit en permettant la correction des données par les autres participants.

3.4. Méthodes applicables pour WE-MAP

Le Tableau 10 présente une synthèse des différentes techniques pour motiver la foule à contribuer dans le cadre d'un *crowdsourcing*. Il en ressort que, l'application ayant un but idéal, nous pouvons compter sur la motivation intrinsèque des utilisateurs que l'altruisme poussera à contribuer. Nous pouvons renforcer cette motivation intrinsèque en proposant un classement des meilleurs contributeurs par territoire et par période.

Si cela ne suffit pas, il pourrait être envisagé de récompenser la participation par la distribution de bons cadeaux de sponsors, par exemple. Cependant il faudrait convaincre ces derniers, ce qui demande un travail supplémentaire.

Tableau 10 – Avantages et inconvénients des différentes méthodes de motivation au crowdsourcing *et utilisabilité de ces méthodes pour WE-MAP.*

Type	Technique	Avantages	Inconvénients	WE-MAP
Participation involontaire		Aucun coût et ne demande pas d'effort apparent à l'utilisateur	Il faut que la tâche demandée à l'utilisateur soit vraiment très simple	Les données demandées à l'utilisateur sont trop complexes pour être saisies à son insu
Motivation intrinsèque	Aspect social, altruisme	Aucun coût et souvent très efficace.		Nous pouvons parfaitement jouer sur l'altruisme des valides et la solidarité des personnes à mobilité réduite
	Ludification	Aucun coût et souvent efficace sur la durée	Le meilleur système dépend des circonstances et n'est pas évident à déterminer	Un système de classement des contributeurs par territoire et par période devrait être efficace
Motivation extrinsèque	Financière	C'est efficace	Coût et tentation de privilégier la quantité plutôt que la qualité des données	Nous n'avons pas de budget pour cela. Un système de bons chez des sponsors serait possible
	Non financière, liée à l'utilisation de l'application	Aucun coût	Rate le groupe des personnes valides	Pas une bonne idée, car les valides ne sont pas incités et les personnes à mobilité réduite ne peuvent pas forcément contribuer

Source : Tableau de l'auteur.

4. Analyse

Quand on développe une application, il est important de savoir comment elle va être acceptée par le public. Nous ne pouvons pas nous contenter de notre propre satisfaction d'avoir développé quelque chose qui fonctionne. Faute de temps, nous nous sommes concentrés sur une analyse qualitative. Il faut tenir compte du fait que notre travail s'inscrit dans un projet plus large de la HES-SO Valais-Wallis qui fera lui-même l'objet d'analyses dans sa partie finale.

4.1. Méthodologie

4.1.1. Focus groups

Nous avons choisi de faire une analyse avec des *focus groups* ou groupes de discussion. Il s'agit d'une discussion en petits groupes, dirigée par un modérateur. Cela permet d'en apprendre plus sur les manières de penser, les convictions, les désirs et les réactions aux concepts des utilisateurs. C'est à l'origine une technique de recherche de marchés. On écoute ce que dit l'utilisateur de ses expériences, mais on n'observe pas directement ses expériences (Digital Communications Division in the U.S. Department of Health and Human Services' (HHS) Office of the Assistant Secretary for Public Affairs, s.d.).

La conduite de la discussion nécessite d'avoir choisi les sujets à aborder à l'avance, de préparer des questions ouvertes pour encourager la discussion, d'organiser l'ordre des questions dans un flux naturel, mais en se gardant la possibilité de changer en cours d'entretien pour que la discussion se déroule tout en douceur. La conversation doit être enregistrée. Idéalement, il faudrait engager un modérateur doué et avoir une ou deux personnes qui prennent des notes (*ibid.*). En raison des moyens à notre disposition, cela ne nous a pas été possible. Une seule personne a animé le groupe et pris des notes durant la séance. Il est d'autant plus important que l'entretien ait été enregistré afin de transcrire ensuite ce qui s'y est dit.

4.1.2. Participants

L'application n'est pas réservée qu'aux personnes handicapées, mais elle est aussi ouverte à leur entourage, ainsi qu'à toutes les personnes prêtes à saisir des informations. Par ailleurs, le but de l'analyse est de tester son utilisabilité. De plus, nous n'avons pas prévu à ce niveau du projet de réunir un échantillon de personnes à mobilité réduite. Les testeurs seront donc des personnes valides.

Deux groupes de testeurs ont été créés. Le premier (G1) comportait quatre étudiants en informatique de gestion de quatrième année à temps partiel. C'étaient donc des personnes jeunes et bien familiarisées avec les nouvelles technologies. Le second groupe (G2)

comportait trois personnes un peu plus âgées, entre 34 et 44 ans. Sans être réfractaires aux nouvelles technologies, elles étaient moins aguerries que les membres du premier groupe.

4.1.3. Procédure

Les participants ont reçu un accès à l'application et un mode d'emploi détaillé pour l'utiliser. Ils ont eu une semaine pour tester l'application dans leurs régions respectives. Il leur a été spécifiquement demandé d'enregistrer des points dans la base de données et de tester des calculs d'itinéraires.

Après une semaine, chaque groupe a été réuni dans une salle avec un testeur. Le lieu de réunion a été choisi pour être agréable et isolé, afin que les participants se sentent parfaitement à l'aise de s'exprimer, sans témoins indiscrets.

En guise d'introduction, nous avons bien sûr remercié les participants pour leur présence et pour le temps consacré aux tests. Nous leur avons rappelé les buts de l'application, puis ceux de l'évaluation à laquelle ils contribuent. Nous avons ensuite insisté sur le fait qu'il n'y a pas de bonne ou de mauvaise réponse aux questions qui peuvent leur être posées, et qu'ils ne doivent pas chercher à faire plaisir à l'animateur de la séance, mais bien exposer librement leur opinion. Nous les avons enfin avertis que la séance allait faire l'objet d'un enregistrement audio, mais que ce qui serait dit resterait anonyme.

Le modérateur disposait d'une liste de questions ouvertes, afin d'animer la séance. Ces questions n'ont pas été d'emblée présentées aux testeurs, mais celles dont les réponses n'étaient pas venues naturellement durant la conversation ont été posées par l'animateur. Elles ont permis de guider le débat. Ces questions portaient sur l'utilisabilité de l'application quant à la navigation, la saisie des points et le calcul d'itinéraires, puis sur l'intérêt de l'application et enfin sur la manière de pousser les gens à contribuer au *crowdsourcing*. Elles figurent en Annexe III.

La séance enregistrée a duré 45 minutes avec le groupe G1 et 41 minutes avec le groupe G2. L'enregistrement a commencé après les paroles de bienvenue et les explications, quand les participants eurent été informés du dit enregistrement.

4.2. Résultats

Afin de respecter l'anonymat des testeurs, les membres du groupe G1 ont été désignés par les noms « P1 » à « P4 » et ceux du groupe G2 par les noms « P5 » à « P7 ».

Les remarques des participants apparaissent dans les chapitres suivants non pas dans l'ordre chronologique dans lequel elles ont été amenées, mais regroupées par chapitres. La transcription chronologique des entretiens fait l'objet de l'Annexe IV. Une synthèse des

réponses sous forme de tableau se trouve en Annexe V, dans lequel figurent aussi quelques commentaires de l'auteur pour chaque point.

4.2.1. Naviguer dans l'application

P4 trouve les trois boutons en bas d'écran peu visibles, mais les autres participants du groupe G1 ne sont pas d'accord. Le groupe G2 n'a rien dit de tel. Les autres participants n'ont pas éprouvé de difficultés particulières pour la navigation.

4.2.2. Ajouter des informations

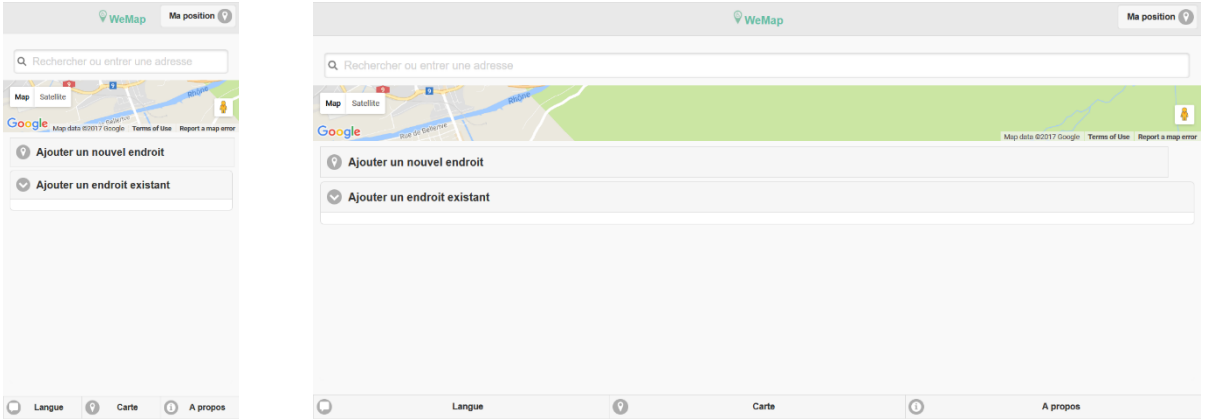
4.2.2.1 Problème de l'activation de la géolocalisation

P3 et P2 ont regretté que si la géolocalisation n'est pas activée pour le navigateur (en particulier sur le téléphone), l'application se bloque et ne signale pas qu'il faut activer la géolocalisation. Le groupe G2 avait été averti d'activer la géolocalisation afin de pouvoir faire les tests.

4.2.2.2 Taille de la carte (position.php)

Globalement, la carte de la page position.php semble petite. La Figure 40 montre cette page telle qu'elle apparaît à gauche sur un écran de smartphone et à droite sur un écran d'ordinateur. Dans le Groupe G1, P3 a immédiatement dit que la carte était trop petite. Selon lui, quand on a sélectionné une adresse, comme la carte est petite et qu'il n'y a pas de zoom, on n'est pas sûr d'être au bon endroit. Le P2 a insisté que surtout sur téléphone, la carte est très petite. P4 trouve que cette petite carte, c'est « frustrant ». P2 souligne qu'il n'y a que deux champs dans la page, qu'il y aurait de la place pour agrandir et que le bas de la page est vide. Dans le groupe G2, P6 a aussi demandé que la carte soit plus grande. P5 estime que le vide en bas ne sert à rien.

Figure 40 – Page position.php*. La carte est considérée comme trop petite par certains participants.*

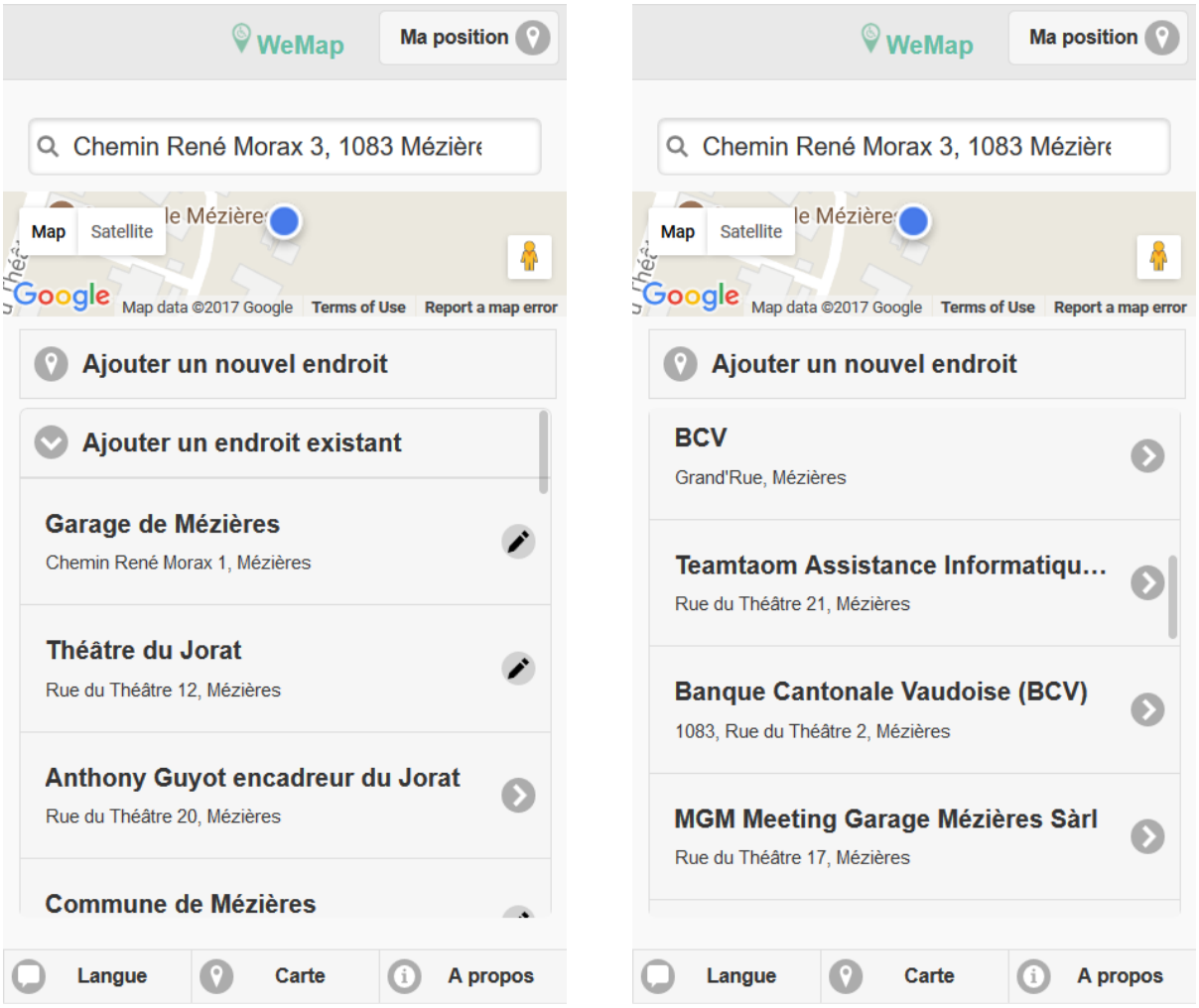


Source : capture d'écran de l'auteur.

4.2.2.3 Ajout d'un endroit existant

P2 a remarqué que quand on scrolle sur la liste qui apparaît sous « Ajouter un endroit existant », le titre « Ajouter un endroit existant » part avec le scroll, comme le montre la Figure 41, sur laquelle on voit qu'il est présent avant le scroll, à gauche, et qu'il ne l'est plus après, à droite. P2 a trouvé que c'est dommage, car on ne sait plus bien où on est.

Figure 41 – Saisie : le titre « Ajouter un endroit existant », présent avant le scroll (à gauche), a disparu lors du scroll (à droite).



Source : capture de l'auteur.

Pour le bouton « Ajouter un point existant », P1 avait compris qu'il s'agissait d'un point existant dans la base de données de WE-MAP, pas dans Google.

P4 souligne qu'il a eu le problème suivant avec la patinoire de L*** : il ne l'a pas trouvée dans les éléments référencés par Google en entrant le nom du village de L*** et l'a créée. Il n'a compris que plus tard que la liste déroulante du bouton « Ajouter un point existant » ne montre que les éléments qui se situent à moins de 200 m (selon lui) du point référencé. En tapant L***, le curseur s'est mis au centre de L*** et la patinoire étant décentrée, elle n'a pas

été proposée. En conséquence, la patinoire est maintenant mal située. P4 regrette qu'on ne puisse déplacer le point après coup.

P3 a souhaité saisir une place dans le village de S***. Cette place n'étant pas référencée dans Google, il a mis juste le nom de la commune de S***. Du coup, l'information est mal située dans le village, vraisemblablement au centre. P3 souhaiterait aussi pouvoir entrer des coordonnées différentes pour un point déjà saisi.

4.2.2.4 Ajout d'un endroit ne remplissant aucun critère

P3 s'est retrouvé confronté à la difficulté de saisir une route qui n'avait aucun des critères à cocher et se demande si la liste des critères va évoluer.

4.2.2.5 Divers

P6 aurait souhaité pouvoir saisir des informations sur les POI en cliquant dessus sur la carte de map.php.

Les participants du groupe G2 voudraient unanimement qu'il y ait un critère déconseillé pour les POI. Il est parfois possible d'aller dans un endroit, mais ce ne sera pas agréable.

P6 et P5 ont souligné l'intérêt des photos qui permettent d'évaluer vraiment le problème en fonction de sa situation.

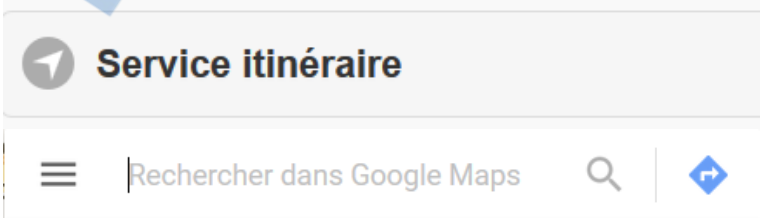
P3 se réjouit que les photos ne soient pas enregistrées sur le téléphone comme le font malheureusement trop d'applications.

4.2.3. Saisie d'itinéraires

P1 a eu du mal à trouver le bouton du calcul d'itinéraire. P3 propose de remplacer l'icône qui le décore, en haut sur la Figure 42, et qui évoque plutôt une boussole ou la géolocalisation par une icône rappelant le calcul d'itinéraires de Google maps, comme en bas sur la Figure 42.

Figure 42 – Icône du calcul d'itinéraire.

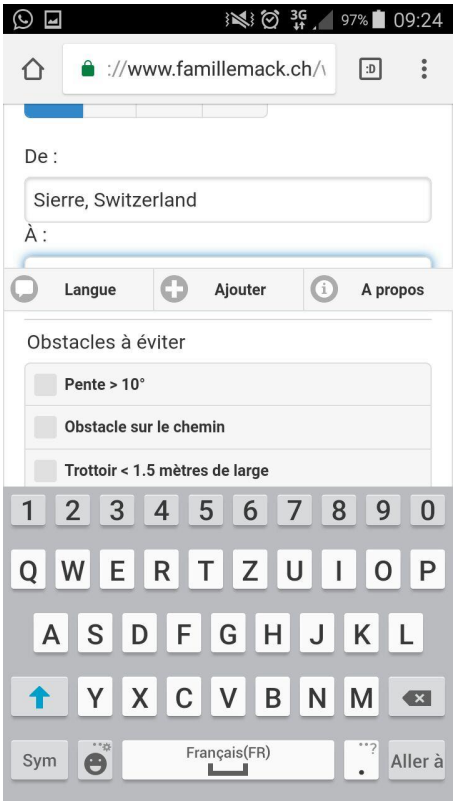
En haut celui de WE-MAP lors des tests des focus groups et en bas (losange bleu à droite) celui de Google maps.



Source : captures d'écran de l'auteur, en haut sur l'application et en bas sur www.maps.google.com .

P2 a remarqué que quand elle saisit un itinéraire pour la deuxième fois, les boutons du bas de l'écran remontent dans le formulaire. Elle transmet une capture d'écran que l'on voit en Figure 43. En effet, les boutons du pied de page sont remontés.

Figure 43 – Capture d'écran d'un participant où l'on voit que les boutons du pied de page remontent.



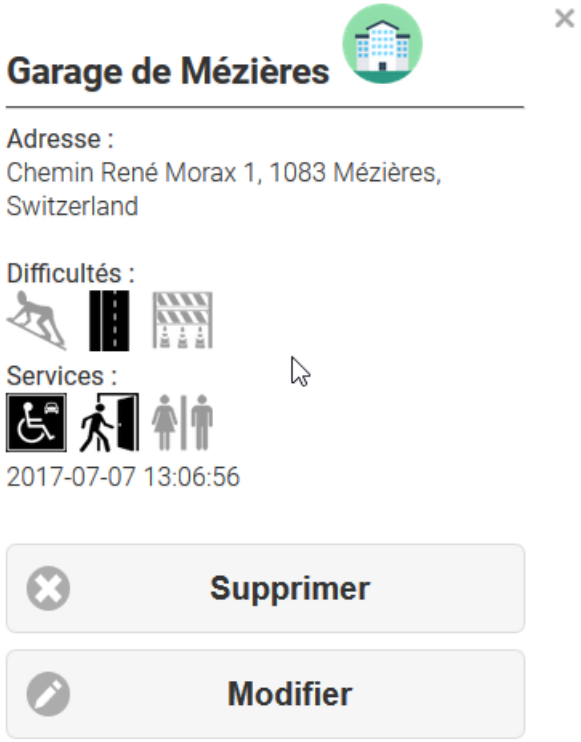
Source : capture d'écran du participant P2.

P4 et P3 auraient souhaité pouvoir calculer un itinéraire qui évite effectivement un obstacle jugé comme infranchissable.

4.2.4. Fenêtre d'information

Les quatre membres du groupe G1 regrettent que les difficultés et services qui ne sont pas présents apparaissent en gris, au lieu d'être simplement absents dans les fenêtres d'information. Ainsi, dans la Figure 44, les difficultés absentes que constituent la pente et l'obstacle et l'absence de toilettes pour handicapés apparaissent en gris. Ils trouvent que ce n'est pas clair. P2 ou P3 pourraient envisager qu'ils soient beaucoup plus transparents. P1 estime qu'il faut les ôter. Pour les membres du groupe G2, la différence entre éléments présents et absents n'est pas nette non plus. P5 propose de barrer les critères absents.

Figure 44 – Fenêtre d'information : exemple.



Source : capture d'écran de l'auteur.

P3 estime que dans cette même fenêtre, le bouton « Supprimer » devrait être en dessous du bouton « Modifier » ou même n'être accessible qu'une fois qu'on a cliqué sur « Modifier ». Il estime que le risque de cliquer sur « Supprimer » par erreur quand on scrolle sur le téléphone est trop grand. P2 abonde dans son sens.

Quand la photo d'un site n'est pas trouvée, il s'affiche un cadre vide, comme le montre la Figure 45. Il serait plus esthétique selon P3 qu'il ne s'affiche rien du tout, comme quand aucune photo n'a été faite. La Figure 45 illustre à gauche une fenêtre pour un point dont la photo est absente et à droite pour un point pour lequel aucune photo n'a été faite.

Figure 45 – Fenêtre d'information : quand l'image n'est pas trouvée, il s'affiche un cadre vide.

C'est ce qu'on voit à gauche. Quand il n'y a pas d'image qui a été saisie, il n'y a pas de cadre vide, comme le montre l'image de droite.

Source : captures de l'auteur.

4.2.5. Carte

Les participants du groupe G1 se demandent si les daltoniens pourront distinguer les points roses et verts. P7 du groupe G2 n'a pas compris la différence entre les points roses et verts et souhaite un mode d'emploi. Les participants des deux groupes ont cité la couleur rose pour les points rouges.

4.2.6. Intérêt de l'application

Les participants du groupe G1 estiment globalement que l'application est utile. P3 qui a saisi une place sur laquelle il est possible de jouer à la pétanque ou de pique-niquer souligne qu'on peut indiquer la présence de gravier tout autour ; la photo permet de se rendre compte si ce gravier pose un problème, pour décider d'aller ailleurs ou pour prévoir du matériel en conséquence. P2 pense à ses grands-parents. Ils ont de la peine à se déplacer. Pour se rendre dans un endroit inconnu, l'application permet de choisir la route à prendre et de déterminer s'ils peuvent y aller seuls.

P4 regrette cependant que l'application ne permette pas de calculer un itinéraire qui contourne vraiment les obstacles.

Les membres du groupe G2 estiment tous que la photo est un plus pour estimer la difficulté. P6, dont la grand-mère était en fauteuil roulant, évoque la nécessité de toujours se renseigner avant d'organiser quoi que ce soit. Elle regrette qu'une telle application n'ait pas existé du vivant de son aïeule.

4.2.7. Crowdsourcing

P2 estime que la gamification peut au mieux encourager les contributeurs qui sont déjà motivés.

P1 propose d'impliquer les employés des offices de tourisme, qui récoltent déjà des données dans les stations durant la basse saison, ainsi que les organisateurs de manifestation.

P7 souligne que les personnes dont cela valorise le local ont intérêt à saisir des données.

4.3. Discussion

4.3.1. Fonctionnalités

Il faut souligner tout d'abord qu'un mode d'emploi a été distribué aux participants une semaine avant l'entretien. Cette opération avait pour but de les guider dans les différentes fonctionnalités à tester, afin qu'ils n'en manquent aucune et que l'analyse soit la plus complète possible. Cependant, cette démarche introduit un certain biais, car il serait préférable que l'utilisation de l'application soit assez explicite pour ne pas nécessiter de mode d'emploi extérieur. Nous avons cependant constaté que certains des participants n'avaient pas complètement lu le tutoriel, puisqu'ils n'ont pas trouvé certaines fonctionnalités qui y étaient décrites.

Par ailleurs, si ces évaluations sont intéressantes, car elles apportent un regard neuf sur le projet au milieu du développement de celui-ci, elles n'ont qu'une faible valeur de preuve, puisqu'il n'y a eu que sept participants. Même si leurs opinions convergent, il faudrait étendre l'échantillon pour généraliser les observations.

Certaines des remarques des participants ont cependant permis d'apporter des modifications dans le cadre de ce travail, en accord avec les responsables du projet. Le détail de ces améliorations se retrouvent au chapitre 4.5.

4.3.1.1 Ajout d'informations

Concernant la nécessité d'activer la géolocalisation sur le téléphone afin de pouvoir cliquer sur « Ma position » sans faire tourner l'application dans le vide, nous avons ajouté, suite à l'entretien, un message qui signale l'erreur à l'utilisateur.

Pour ce qui est de l'ajout d'informations, les participants ont regretté que la carte soit trop petite et que le titre « Ajout d'un point existant » disparaisse lors du scrolling. Ces points ont été fixés ainsi lors du développement antérieur, suite aux tests effectués par les facteurs de Sierre, Sion et du Val d'Anniviers, pour laisser un maximum de place à la liste déroulante. Sur les écrans de petits téléphones, seuls trois éléments apparaissent dans la liste et il y en aurait moins si de la place était perdue pour une carte plus grande ou pour laisser ce titre. Quant à la limite de 200 m pour les POI autour du curseur, elle a été choisie pour éviter que la liste ne comporte des dizaines de points et que la recherche ne soit fastidieuse, et ce à la demande des facteurs. Le nom du bouton « Ajouter un nouvel endroit », qui a été jugé peu clair, était « Ajouter un POI » dans sa version précédente et cela avait été jugé encore moins clair par les employés de la poste.

L'ajout d'un endroit ne remplissant aucun des critères a été rendu entretemps possible par un autre membre de l'équipe. Nous ne reviendrons donc pas sur ce point ici.

Le fait qu'un endroit soit accessible mais plutôt déconseillé peut faire l'objet d'un commentaire et ne nécessite donc aucune adaptation.

4.3.1.2 Saisie et calcul d'itinéraires

Nous avons changé l'icône du bouton de calcul d'itinéraires qui correspond dorénavant à celui de Google Maps, afin que sa fonction soit plus évidente pour des utilisateurs habitués à cet outil.

4.3.1.3 Fenêtres d'information

Dans les fenêtres d'information, nous avons éclairci les icônes des propriétés absentes afin de rendre plus évident le contraste avec les éléments présents. Nous avons aussi ajouté une explication qui s'ouvre quand on clique sur ces icônes. Nous avons enfin inversé l'ordre des boutons « Modifier » et « Supprimer », afin que ce dernier apparaisse en-dessous.

4.3.1.4 Carte

Les participants n'avaient pas compris la différence entre les points rouges et les points verts. De plus, la couleur des points peut varier. En effet, quand on arrive sur la carte, les points rouges représentent les POI qui ont au moins un élément négatif (pente trop raide, trottoir trop étroit ou obstacle), alors que tous les autres sont en vert. Mais quand on calcule un itinéraire n'apparaissent en rouge que les éléments qui ont un des points négatifs qui ont

été sélectionnés par l'utilisateur. Nous avons donc ajouté un petit **tutoriel** afin que la navigation soit plus aisée. Ce tutoriel pourra par la suite être développé pour ajouter d'autres points.

4.3.2. Intérêt de l'application et *crowdsourcing*

Bien que les participants n'aient pas été sélectionnés selon un critère de cet ordre, deux sur sept ont spontanément évoqué avec enthousiasme l'utilité de l'application pour leurs grand-parents. Il n'est évidemment pas question de tirer une conclusion statistique sur un si petit échantillon, mais cela nous incite à penser que l'application pourrait trouver son public. Aucun participant n'a émis la moindre réserve quant à l'intérêt de la plateforme. Un des éléments positifs qui a été plusieurs fois cité est la possibilité de mettre une photo qui décrit la difficulté mieux que des mots.

Au contraire, les participants étaient beaucoup plus réservés sur la possibilité de convaincre les gens d'alimenter la base de données. Il n'est pas évident pour eux que des gens non concernés puissent s'impliquer.

4.4. Conclusion de l'évaluation

Notre évaluation, avant tout qualitative, a été effectuée sur un échantillon relativement petit puisque seules sept personnes y ont participé. On ne peut donc pas en tirer de conclusion définitive. Elle a cependant offert ce que peuvent apporter des regards neufs à ceux qui sont très impliqués dans un projet.

Cette analyse a permis, sur le plan pratique, de détecter quelques défauts de l'application auxquels il a pu être remédié par des modifications mineures que nous avons pu effectuer dans le cadre de ce travail. Elle a aussi permis de souligner quelques aspects qui pourront intéresser les responsables du projet global, sans remettre en cause ce dernier.

L'évaluation a confirmé l'intérêt de l'application, mais aussi la difficulté de motiver la foule à contribuer à la collecte des données.

4.5. Modifications apportées à l'application

Suite à notre évaluation et en accord avec les responsables du projet, nous avons décidé d'apporter quelques améliorations à la plateforme. Nous avons dû nous limiter à ce qui était possible dans le temps imparti à ce travail de Bachelor.

4.5.1. Activation de la géolocalisation

Lorsque l'utilisateur cliquait sur le bouton « Ma position » de l'écran de saisie d'un nouveau POI, si la géolocalisation était inactivée dans le navigateur, il ne se passait rien de visible et aucun message ne s'affichait. L'utilisateur pouvait penser que l'application était plantée et, surtout, il ne savait pas pourquoi.

Ce problème n'est pas présent avec tous les navigateurs. Par exemple avec Firefox, si la géolocalisation est inactivée (c'est-à-dire que si dans les paramètres qu'on atteint avec « about:config » le paramètre « geo.enabled » est à *false*), le navigateur ouvre une fenêtre d'avertissement avec le message : « Your browser doesn't support HTML5 geocoding ». En effet, les développeurs qui ont exécuté la première partie de l'application ont bien mis un test

```
if (navigator.geolocation)
```

qui, s'il est faux, affiche au moyen d'un alert() le message ci-dessus. Malheureusement selon les navigateurs, le test est vrai mais la méthode getCurrentLocation() retourne ensuite une erreur.

Nous avons choisi de remédier à cela en affichant un message d'erreur. Comme parfois les navigateurs bloquent les fenêtres *pop-up*, nous avons choisi d'afficher le message comme dans un *toast* d'Android, c'est-à-dire que le message s'affiche quelques secondes dans une fenêtre au milieu de l'écran et disparaît ensuite. Non seulement cela contourne les filtres à *pop-up*, mais en plus c'est élégant. Pour afficher un *toast*, nous avons ajouté dans le fichier position.js (qui contient le script lié à la page position.php) la fonction de la Figure 46.

Figure 46 – Fonction toast(). Simule un toast d'Android.

```
/**
 * Android-like toast
 * David Mack adapted from https://gist.github.com/kamranzafar/3136584
 *
 * This function display the message msg in a window during 1.5 seconds.
 * @param msg (String) Message to display
 */
function toast(msg) {
    $("<div class='ui-loader ui-overlay-shadow ui-body-e
                ui-corner-all'><h3>"+msg+"</h3></div>")
        .css({ display: "block",
            opacity: 0.90,
            position: "fixed",
            padding: "7px",
            "text-align": "center",
            width: "270px",
            left: ($(window).width() - 284)/2,
            top: $(window).height()/2 })
        .appendTo( $.mobile.pageContainer ).delay( 1500 )
        .fadeOut( 400, function(){
            $(this).remove();
        });
};
```

Source : code de l'auteur d'après Kamranzafar (2012)

Lorsqu'on clique sur le bouton « Ma position », c'est la fonction getCurrentPosition(), dans le fichier position.js, qui est exécutée. Cette fonction utilise la fonction

navigator.geolocation.getCurrentPosition() pour récupérer les coordonnées de la position de l'utilisateur. La syntaxe de cette dernière fonction est :

navigator.geolocation.getCurrentPosition(*success*[, *error*[, *options*]])

où *success* est la fonction exécutée en cas de succès, dont le paramètre est l'objet Position ; *error* est la fonction exécutée en cas d'erreur, dont le paramètre est l'objet PositionError ; et enfin *options* est un objet PositionOptions. Seul le premier paramètre, *success*, est obligatoire (Mozilla Foundation, s.d.).

Nous avons donc ajouté la fonction de la Figure 47 en second paramètre. Ainsi l'utilisateur reçoit-il au moins une information quand la géolocalisation ne se fait pas.

Figure 47 – Fonction exécutée lorsque navigator.geolocation.getCurrentPosition() retourne une erreur.

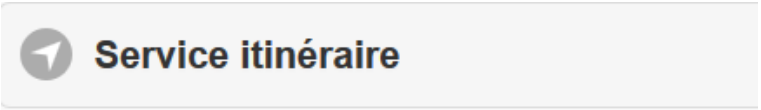
```
function (error) {
    // 2nd argument of getCurrentPosition = function executed if an error is
                                                                    raised

    $.mobile.loading('hide');
    switch(error.code) {
        case error.PERMISSION_DENIED:
            toast("User denied the request for Geolocation. Or Geolocation
                                is not activated in browser.");

            break;
        case error.POSITION_UNAVAILABLE:
            toast("Location information is unavailable.");
            break;
        case error.TIMEOUT:
            toast("The request to get user location timed out.");
            break;
        case error.UNKNOWN_ERROR:
            toast("An unknown error occurred.");
            break;
        default:
            toast("An other unknown error occured");
    }
}
```

Source : code de l'auteur.

4.5.2. Calcul d'itinéraire

Les participants n'ont pas trouvé facilement le bouton permettant de calculer l'itinéraire. Ils ont en particulier remarqué que l'icône choisie (Figure 48) ne faisait pas penser à un calcul d'itinéraire, mais plutôt à une géolocalisation. Pour les personnes habituées à travailler avec Google Maps, l'icône de calcul d'itinéraires ressemble à un losange avec une flèche à angle droit qui indique la droite.

Figure 48 – Bouton de calcul de l'itinéraire avec l'icône de géolocalisation.



Source : capture de l'auteur.

L'icône utilisée est fournie par le *framework* JQuery Mobile utilisé pour l'application. Le problème est que ce *framework* ne met à disposition qu'une cinquantaine d'icônes. La librairie *Font Awesome* que nous utilisons aussi dans l'application n'a pas non plus une telle icône. Il est possible d'utiliser directement l'icône de Google, plus précisément celle qui est utilisée pour afficher le résultat des itinéraires avec l'instruction : « Prendre à droite ». Pour cela, il faut d'abord insérer dans la balise <head> un lien vers la feuille de style de Google Maps :

<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">

puis insérer une balise , par exemple :

directions

à l'endroit où on souhaite afficher l'icône. Cette solution est possible mais elle présente des inconvénients. L'icône ainsi tracée ne sera qu'un élément du texte du bouton, elle ne prendra pas place dans le petit rond à gauche du bouton. Des effets de style permettent de tracer un rond grisé autour, mais l'aspect sera différent, ce qui est en rupture avec l'esthétique de l'application. De plus, l'icône ne sera pas alignée horizontalement avec le texte, ce qui est inesthétique.

Nous avons donc choisi une autre solution. Nous avons trouvé cet icône en libre accès sur www.materialui.co/icon/directions. Nous l'avons copiée dans notre projet. Puis nous l'avons référencée dans le code CSS selon la Figure 49.

Figure 49 – Style de la page map.php dans le fichier map.css : référence à l'icône directions.

```
.ui-icon-direction:after {
    /* https://www.materialui.co/icon/directions */
    background-image: url("images/directions_white_192x192.png");
    background-size: 18px 18px;
}
```

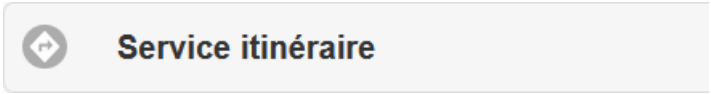
Source : code de l'auteur.

Nous avons enfin ajouté le code :

```
<span class="ui-btn-icon-left ui-icon-direction"></span>
```

juste avant le texte du bouton et le résultat se trouve en Figure 50.

Figure 50 – Bouton de calcul de l'itinéraire avec la nouvelle icône de directions.



Source : capture de l'auteur.

4.5.3. Fenêtres d'information : grisé des critères absents

Nous avons augmenté la transparence en faisant passer l'opacité de 40% à 20%. Cette ligne de code devient donc :

```
infotext += "opacity: 0.2;filter: alpha(opacity=20);";
```

4.5.4. Fenêtres d'information : *tooltips* sur les critères

Les icônes des critères n'étaient pas forcément claires pour les testeurs. Par exemple, si l'on se rappelle que la pente pose un problème, c'est à partir de combien de degrés ? Un des participants a proposé que l'on mette une petite bulle d'information quand on clique sur une des icônes sur le téléphone ou quand on passe le pointeur de la souris au-dessus sur l'ordinateur. Nous avons donc décider d'ajouter des *tooltips*.

Dans un premier temps, en nous inspirant de w3schools (s.d.), nous avons créé un fichier tooltips.css dont le contenu est en Figure 51. Comme les icônes sont à gauche de la fenêtre d'information, nous avons dû faire quelques adaptations pour qu'il s'ouvre décentré par rapport à l'icône. C'est pourquoi dans sa position le *tooltip* a une marge de gauche (margin-left) de -15 px et que la flèche est à 13% ($\cong \frac{15}{120}$) depuis la marge de gauche.

Nous n'avions plus qu'à mettre chaque image dans un *tooltip container* et à lui attribuer un *tooltip text*. La Figure 52 montre ce code modifié dans le script de map.php.

Figure 51 – Contenu du fichier tooltip.css qui définit les tooltips.

```css
/* David Mack adapted from
        https://www.w3schools.com/howto/howto_css_tooltip.asp - Tooltips */
/* Tool tip container */
.tooltip {
    position: relative;
    display: inline;
    width: 30px;
    height: 30px;
}

/* Tooltip text */
.tooltip .tooltiptext {
    visibility: hidden;
    width: 120px;
    background-color: #555;
    color: #fff;
    text-align: center;
    border-radius: 6px;
    padding: 5px 0;

    /* Position the tooltip text */
    position: absolute;
    z-index: 1;
    bottom: 200%;
    left: 50%;
    margin-left: -15px;

    /* Fade in tool tip */
    opacity: 0;
    transition: opacity 1s;
}

/* Tooltip arrow */
.tooltip .tooltiptext::after {
    content: "";
    position: absolute;
    top: 100%;
    left: 13%;
    margin-left: -5px;
    border-width: 5px;
    border-style: solid;
    border-color: #555 transparent transparent transparent;
}

/* Show the tooltip text when you mouse over the tooltip container */
.tooltip:hover .tooltiptext {
    visibility: visible;
    opacity: 1;
}
```

Source : code de l'auteur (tooltip.css), d'après w3schools (s.d.).

Figure 52 – Ajout du tooltip dans le script de map.php, dans le fichier map.js.

```
// Put the tooltip container
infotext += "<div class='tooltip' id='id_"+ path_type +"'>";
// Put the tooltip text
infotext += "<span class='tooltiptext'>";
infotext += titles[path_type];
infotext += "</span>";
// Put the image
infotext += "<img src='images/" + path_type + ".png' alt='" + path_type + "'
style='width:30px;height:30px;";
// if the criterion is not present, the image will be transparent
if(!path_types[path_type]) {
    infotext += "opacity: 0.2;filter: alpha(opacity=20);";
}
infotext += "'>";
infotext += "</div>"; // end of tooltip container
```

Source : code de l'auteur (map.js).

4.5.5. Fenêtres d'information : ordre des boutons « Modifier » et « Supprimer »

Comme un des participants aux *focus groups* a souligné le risque de cliquer sur le bouton « Supprimer » quand on scrolle pour chercher le bouton « Modifier », nous avons inversé l'ordre de ces boutons. Cela n'a présenté aucune difficulté particulière, il suffisait d'inverser l'ordre des deux lignes concernées.

4.5.6. Tutoriel

Les testeurs ayant eu parfois des difficultés à utiliser l'application, malgré le mode d'emploi qui leur avait été distribué une semaine à l'avance, nous avons ajouté un tutoriel qui comporte pour l'instant cinq écrans accompagnés d'un commentaire. Ce tutoriel est géré dans un fichier tutorial.php auquel l'utilisateur accède sur le premier écran, celui du choix de la langue, comme le montre la Figure 53.

Dans le fichier tutorial.php, nous avons au centre deux superposées, l'une pour l'illustration et l'autre pour le texte. En bas, trois boutons permettent de reculer dans le tutoriel, de le quitter ou d'avancer.

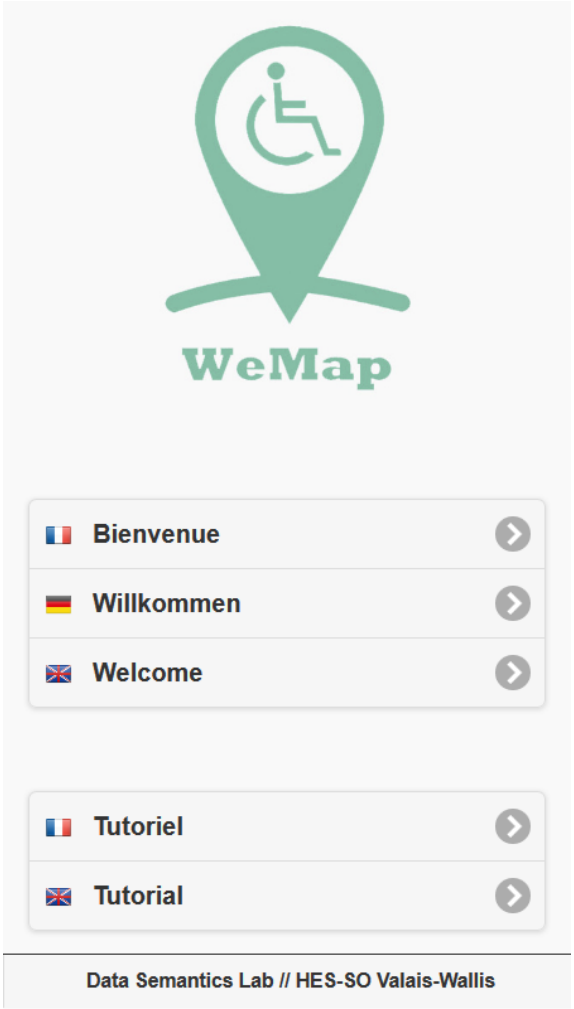
Afin qu'il soit facile d'ajouter des pages si de nouvelles fonctionnalités devaient être décrites, nous avons procédé de manière systématique. Nous avons mis les étapes du tutoriel dans un tableau :

```
var tutorial_flow = ['POSITION', 'COLLECT', 'MAP', 'ITINERARY_CALCULATE',
                     'ITINERARY_RESULT'];
```

Les textes du tutoriel ont été mis dans les fichiers de langue lang.fr.php et lang.en.php dans des variables qui portent les mêmes noms que les étapes. Et les illustrations ont des noms

dérivés du nom des étapes et de la langue, par exemple tutorial_POSITION_fr.png. Les textes et les illustrations sont chargées dans deux tableaux, respectivement texts[ ] et pictures[ ].

Figure 53 – Écran initial avec les tutoriels en français et anglais.



Source : capture de l'auteur.

La navigation repose ensuite sur les fonctions assez simples de la Figure 54.

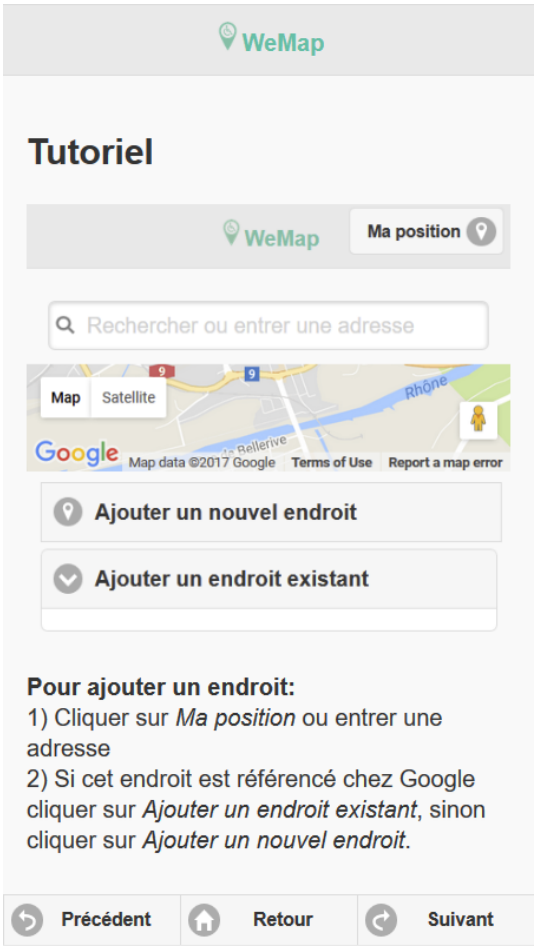
Figure 54 – Tutoriel : fonctions de navigation.

```
function setStep(step) {
    illustration.src = pictures[tutorial_flow[step]];
    tuto_text.innerHTML = texts[tutorial_flow[step]];
}

function nextStep() {
    current_step ++;
    if (current_step >= tutorial_flow.length) {
        current_step = 0;
    }
    setStep(current_step);
}

function previousStep() {
    current_step --;
    if (current_step <= -1) {
        current_step = tutorial_flow.length - 1;
    }
    setStep(current_step);
}
```

Source : code de l'auteur.

La Figure 55 illustre un exemple d'étape du tutoriel.

Figure 55 – Tutoriel : exemple de présentation.



Source : capture de l'auteur.

5. Gestion du projet

5.1. Planification et cahier des charges

La planification du projet s'est faite principalement durant les trois premières semaines. Le cahier des charges qui en est ressorti figure en Annexe I de ce mémoire. Le Tableau 11 présente la liste des phases du travail et la part du temps qu'il a été prévu de lui consacrer.

Tableau 11 – Planification : répartition du temps entre les phases du projet.

	Phase	Heures	Pourcentage du temps
Phase 0	Planification	33	9.2 %
Phase 1	État de l'art du calcul d'itinéraire	55	15.3 %
Phase 2	Développement	116	32.2 %
Phase 3	État de l'art du *crowdsourcing*	48	13.3 %
Phase 4	Évaluation de l'application	48	13.3 %
Phase 5	Rédaction	60	16.7 %
	Totaux	**360**	**100.0 %**

Source : cahier des charges du travail.

5.2. Rencontres avec les responsables du projet

Durant ce travail, nous avons rencontré les responsables du projet toutes les 2 à 4 semaines, en fonction des besoins et des disponibilités de chacun. Ainsi, si nous n'avons pas fait formellement du *scrum*, nous avons néanmoins gardé cet aspect de la rencontre régulière qui permet de valider ce qui a été réalisé, comme un *sprint review* et de préparer ce qui va être accompli jusqu'à la prochaine rencontre, comme un *sprint planning*. Pour ces rencontres, nous avons pu profiter de l'implication de chacun. Le Tableau 12 donne le détail de ces rencontres.

Le jeudi 18 mai 2017, l'objectif du calcul d'itinéraire a un peu évolué par rapport à ce qui avait été écrit dans le cahier des charges (Annexe I, chapitre 7.2). Madame Nicole Glassey Balet nous a précisé qu'elle ne souhaitait pas que l'itinéraire contourne les obstacles mais qu'il les signale. En effet, contourner un obstacle peut impliquer un grand détour et l'utilisateur peut préférer l'affronter.

Tableau 12 – Rencontres avec les responsables du projet.

Date	Présence de Nicole Glassey	Présence de Zhan Liu	Objet de la séance
Je 23.02.2017 16h00	✓	✓	Mise en route.
Je 16.03.2017 15h30	✓	✓	Finalisation du cahier des charges.
Ve 31.03.2017 15h30	✓	✓	Présentation de l'avancement de l'état de l'art sur les itinéraires.
Je 27.04.2017 15h30	✓		Discussion de l'état de l'art sur les itinéraires.
Me 03.05.2017 09h00 Ecublens		✓	Mise en route de l'implémentation. Choix de l'outil (Google Maps)
Je 18.05.2017 16h00	✓	✓	Présentation du calcul Précisions quant à la présentation
Ve 19.05.2017 15h45		✓	Échange de code
Ma 13.06.2017 09h00	✓	✓	Présentation calcul itinéraire. Décisions de modifications. Décisions concernant la modification des InfoWindows. Décision de procéder à l'évaluation par *focus groups*.
Ve 23.06.2017 10h30		✓	Dernières modifications
Lu 10.07.2017 10h00	✓	✓	Présentation de l'état de l'art sur le *crowdsourcing*. Premiers résultats de l'évaluation.
Lu 10.07.2017 (suite)		✓	Décisions quant aux modifications à apporter suite à l'évaluation.

Source : données de l'auteur.

5.3. Gestion du temps

Dans sa documentation, l'école a proposé que les étudiants consacrent 11 heures hebdomadaires au travail de diplôme durant les neuf premières semaines, du 20 février au 30 avril 2017, sachant que la semaine du 17 au 23 avril est une semaine de vacances ; puis 15 heures hebdomadaires du 1er mai au 18 juin 2017 ; puis 24 heures hebdomadaires du 19 juin au 6 août, avec une semaine consacrées aux examens du 26 juin au 2 juillet 2017 ; et enfin 12 heures la semaine du 7 au 13 août, le délai de remise étant fixé au 9 août (Tacchini, 2016). Le Tableau 13 illustre cette répartition des 360 heures sur théoriquement 23 semaines consacrées au travail de Bachelor durant 25 semaines.

Tableau 13 – Répartition des heures consacrées au travail selon la planification de l'école.

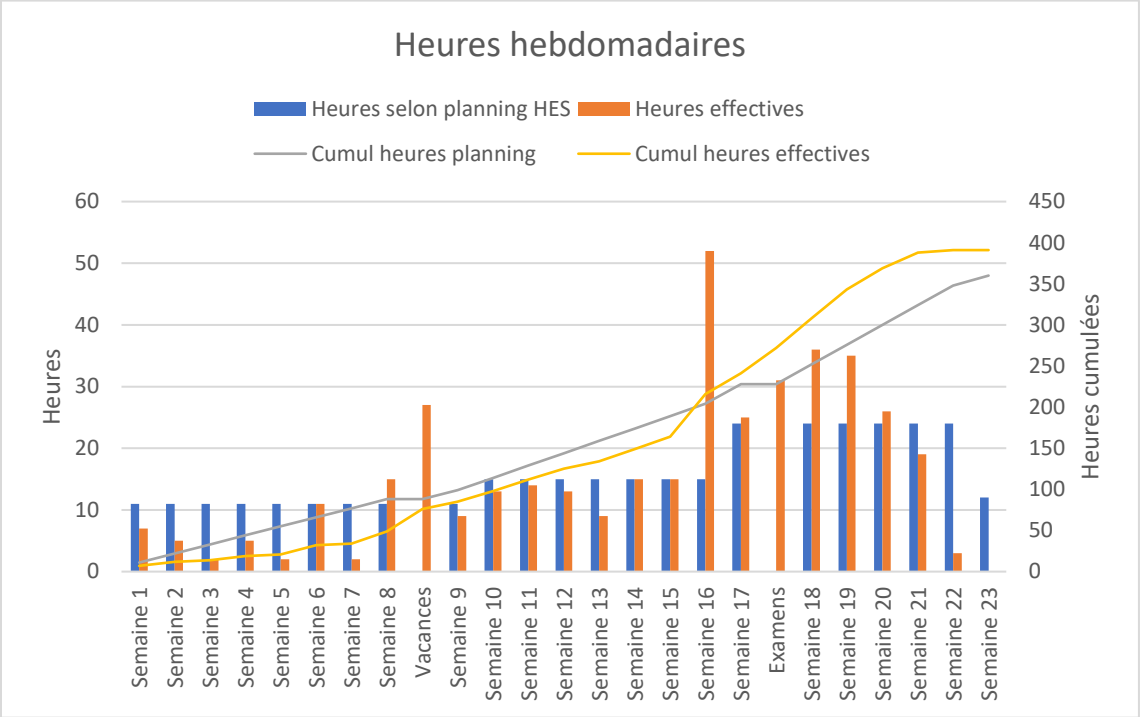
Semaine	Lundi	Dimanche	Heures selon HES
Semaine 1	20.02.2017	26.02.2017	11
Semaine 2	27.02.2017	05.03.2017	11
Semaine 3	06.03.2017	12.03.2017	11
Semaine 4	13.03.2017	19.03.2017	11
Semaine 5	20.03.2017	26.03.2017	11
Semaine 6	27.03.2017	02.04.2017	11
Semaine 7	03.04.2017	09.04.2017	11
Semaine 8	10.04.2017	16.04.2017	11
Vacances	17.04.2017	23.04.2017	0
Semaine 9	24.04.2017	30.04.2017	11
Semaine 10	01.05.2017	07.05.2017	15
Semaine 11	08.05.2017	14.05.2017	15
Semaine 12	15.05.2017	21.05.2017	15
Semaine 13	22.05.2017	28.05.2017	15
Semaine 14	29.05.2017	04.06.2017	15
Semaine 15	05.06.2017	11.06.2017	15
Semaine 16	12.06.2017	18.06.2017	15
Semaine 17	19.06.2017	25.06.2017	24
Examens	26.06.2017	02.07.2017	0
Semaine 18	03.07.2017	09.07.2017	24
Semaine 19	10.07.2017	16.07.2017	24
Semaine 20	17.07.2017	23.07.2017	24
Semaine 21	24.07.2017	30.07.2017	24
Semaine 22	31.07.2017	06.08.2017	24
Semaine 23	07.08.2017	13.08.2017	12
		Total	**360**

Source : support de la séance d'information (Tacchini, 2016)

Dans les faits, nous n'avons pas arrêté de travailler durant la semaine des vacances de Pâques ni dans celle d'examen. Bien au contraire, nous avons profité de ces périodes pour avancer dans la tâche. La Figure 56 donne la répartition des heures hebdomadaires effectives en comparaison avec celles de la planification de l'école. On voit que du retard a surtout été pris durant les trois premières semaines, qui cumulent 14 heures effectives. Ces semaines étaient consacrées à la planification et cette dernière a nécessité moins que les 33 heures prévues par le planning de l'école pour cette même période. De plus, ce temps est sans doute sous-évalué car du temps libre a déjà été consacré à des premières recherches générales sur le calcul d'itinéraire. Durant les deux semaines qui ont suivi, le nombre d'heures effectuées est très bas en raison d'obligations professionnelles. Le retard pris durant les cinq premières semaines a été quasiment rattrapé durant les vacances de Pâques. Le pic de la semaine 16 est dû à une semaine de vacances prise au travail et consacrée à ce projet. De l'avance a

encore été prise durant la semaine « d'examens », ces derniers ayant eu lieu le vendredi de la semaine précédente. Nous avons dès le début renoncé à utiliser la semaine 23. Le planning de l'école prévoyait de consacrer 12 heures au travail de Bachelor cette semaine-là. Or le délai de remise est fixé au mercredi à midi. Travaillant à plein temps, il ne nous aurait pas été possible de trouver autant d'heures entre lundi et mardi.

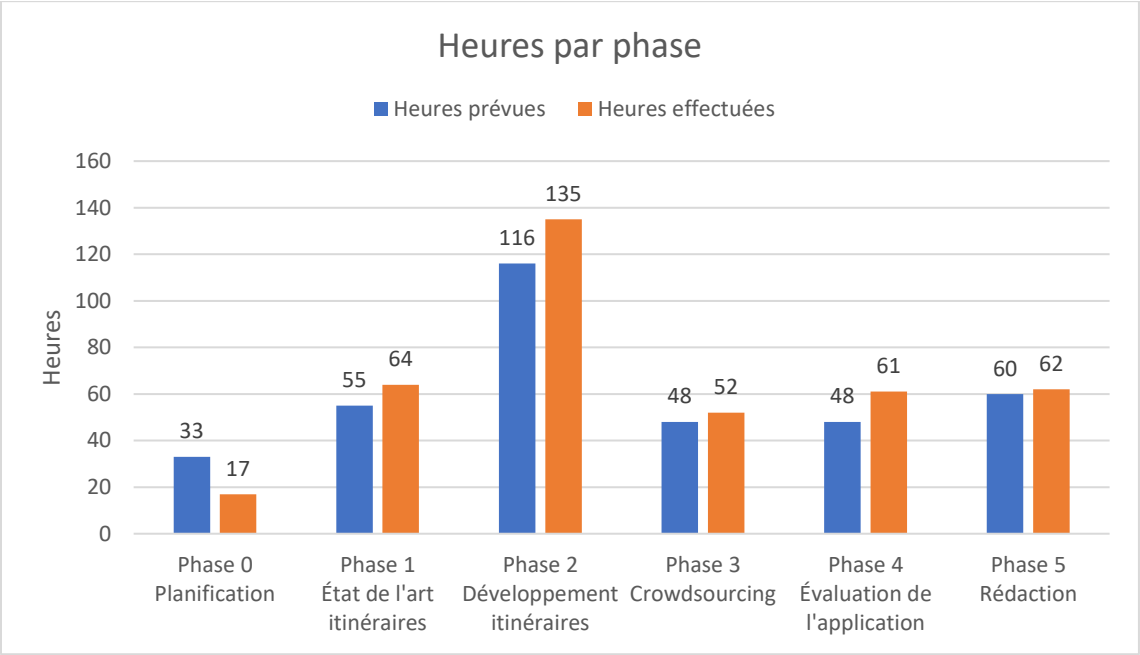
Figure 56 – Comparaison des heures hebdomadaires selon le planning de l'école et des heures effectives.



Source : chiffres de l'auteur.

La Figure 57 illustre la répartition des heures entre les différentes phases du travail. Le nombre total a été de 391 heures. Comme nous l'avons dit au chapitre précédent, le nombre d'heures planifiées était lié aux semaines et au nombre d'heures que l'école prévoyait pour le travail de Bachelor. La phase de planification a donc pris moins d'heures que prévu. L'état de l'art des itinéraires a pris 9 heures de plus que prévu, soit 16 %. Le développement a pris 19 heures de plus que prévu, soit 16 %. La recherche sur le *crowdsourcing* a pris 4 heures de plus que prévu, soit 8 %. La phase d'évaluation a pris 13 heures de plus que prévu, soit 27 %, mais il faut noter que nous avons profité de cette phase pour améliorer l'application. Ce temps n'a donc pas été consacré qu'à l'analyse elle-même. La phase de rédaction a pris 2 heures de plus que le temps prévu. Il faut bien sûr remarquer qu'une part importante de la rédaction avait été réalisée durant le travail, au fur et à mesure. La phase 5 a donc surtout servi à rédiger les parties initiales et finales, à mettre en forme et à relire le mémoire.

Figure 57 – Comparaison entre les heures consacrées à chaque phase selon la planification et celles qui ont été effectuées.



Source : chiffres de l'auteur

5.4. Respect des délais

Le Tableau 14 rappelle les délais qui avaient été convenus lors de l'établissement du cahier des charges et indique les dates auxquelles les éléments ont effectivement été rendus.

Tableau 14 – Respect des délais : comparaison des dates prévues et des dates effectives.

Échéance	Date prévue	Date effective
Début du projet	20.02.2017	23.02.2017
Remise cahier des charges	15.03.2017	11.03.2017
État de l'art calcul itinéraires	13.04.2017	26.04.2017
Calcul d'itinéraires et autres développement	15.06.2017	30.06.2017
État de l'art *crowdsourcing*	06.07.2017	09.07.2017
Évaluation de l'application	20.07.2017	10.07.2017 (1ers éléments) 20.07.2017 (v. finale)
Remise du travail de diplôme	09.08.2017	03.08.2017

Source : cahier des charges et données de l'auteur.

Des différences de quelques jours ne sont pas significatives, car la remise des éléments était fonction des rencontres avec les responsables du projet dont les dates dépendaient des agendas de chacun. Nous voyons que deux semaines de retard environ ont été prises au début du projet, pour l'état de l'art du calcul d'itinéraires et son développement. Nous

soulignons cependant que la phase de développement est allée au-delà du seul calcul d'itinéraire. Ce retard s'est ensuite résorbé durant les phases suivantes.

5.5. Outils utilisés

L'application internet préexistante était en HTML5, PHP, JavaScript et MySQL. Elle utilisait la bibliothèque JQuery et le *framework* JQuery Mobile, qui gère les interfaces pour des appareils mobiles. Nous avons donc logiquement repris ces langages et bibliothèques.

Pour ce qui est des cartes et du calcul d'itinéraires, l'application utilisait l'API Google Maps Javascript. C'est donc très logiquement et pour garder la cohérence que nous avons choisi de continuer avec cette API.

Comme outil de développement, nous avons utilisé PhpStorm de JetBrains. Nous avons choisi cet outil car, d'une part, il est gratuit pour les étudiants et, d'autre part, nous avons l'habitude de travailler avec des outils de JetBrains, puisque nous avons eu l'occasion durant notre formation d'utiliser aussi Android Studio, PyCharm et WebStorm.

Pour la gestion des versions, nous avons utilisé Bitbucket. Cette plateforme offre l'avantage que jusqu'à cinq développeurs, un répertoire privé est disponible gratuitement, alors que sur d'autres plateformes, il faut payer pour en disposer. Sinon, le code est librement accessible à tout le monde. Il faut remarquer que cette plateforme a été utilisée uniquement dans le cadre de ce travail de Bachelor, avec donc un seul développeur. Elle n'a donc pas servi à partager le code, mais uniquement à gérer et sauvegarder les versions.

Le décompte des heures s'est fait dans les rapports hebdomadaires rédigés selon le modèle fourni par l'école et dont l'intégralité se trouve sur le CD-ROM qui accompagne la version imprimée de ce mémoire. Le détail des heures a ensuite été récapitulé dans un tableau Excel.

6. Bilan

Ce travail de diplôme a permis à l'auteur de gérer seul un projet du début à la fin. Bien que ce projet fît partie d'un ensemble plus vaste, puisqu'une application avait déjà été développée auparavant et qu'il y aura encore des développements ultérieurs, un cahier des charges a été établi pour ce travail et il a été possible de le respecter. C'est toujours satisfaisant d'atteindre des objectifs qui ont été fixés. Après une recherche sur l'état de l'art et une importante partie de développement, le fait d'analyser le résultat, même de manière succincte, et de tirer des conclusions de cette analyse permet de terminer l'aventure sans la laisser inachevée.

De plus, le projet WE-MAP dans le cadre duquel nous avons travaillé est lui-même très motivant, car il vise à aider des personnes à mobilité réduite. Nous espérons que l'application pourra effectivement être mise à disposition du public pour une large diffusion. Dans tous les cas, notre travail devrait être utile aux responsables du projet global et c'est gratifiant.

Certaines des techniques utilisées dans le développement étaient complètement neuves pour l'auteur. Durant la formation, il n'avait jamais été confronté directement à Google Maps Javascript API. Plus généralement, le JavaScript n'avait pas été étudié tout au long de la formation et il a fallu fournir un important exercice de formation en plus du reste.

Ce travail a comporté une importante masse de travail à accomplir. Il s'est fait, d'une part, en parallèle avec les cours, les projets et les examens pour ce qui est de l'école et, d'autre part, en même temps que les obligations professionnelles à 80 pour cent et même 100 pour cent depuis le 17 juillet, associatives et familiales. C'était un défi que nous pensons avoir relevé avec succès. Mais cela n'a pas été sans sacrifices tant pour la vie sociale que pour les heures de sommeil.

Nous regrettons cependant de ne pas avoir pris plus de temps dès le début, par exemple en prenant congé au travail, pour tenir mieux les délais. Nous aurions peut-être aussi dû anticiper en nous confrontant plus précocement au JavaScript et à ses *frameworks*, afin d'être plus efficace durant la phase de développement.

L'auteur a globalement apprécié cette première grande expérience professionnelle de projet d'informatique de gestion qui lui a permis de mettre en pratique ce qu'il a appris durant ces quatre années, en particulier de se débrouiller avec des techniques inconnues, le tout dans un climat de travail très agréable.

Conclusion

Dans la **première partie** de ce travail, nous avons recensé plusieurs applications qui calculent des itinéraires pour des personnes à mobilité réduite. Nous avons cité Handimap, qui permet le calcul d'itinéraires pour différents types de handicap à Rennes, Montpellier et dans l'agglomération de Lorient. Cette plateforme donne quelques informations, par exemple les places de parc adaptées, à la Rochelle et à Nice. En Allemagne, l'Université de Heidelberg propose aussi un site (Rollstuhlrouting) qui permet des calculs d'itinéraires pour les handicapés. Mais les possibilités restent limitées, aussi à Heidelberg-même. Dans le même pays, Wheelmap répertorie sur une carte d'OpenStreetMap les sites et les toilettes accessibles en fauteuil roulant. Toutes ces applications sont incomplètes. Soit elles se limitent à une zone géographique restreinte, comme Handimap, soit elles permettent le calcul d'itinéraires mais ne fournissent aucune donnée complémentaire sur les services, comme la solution de l'Université de Heidelberg, soit au contraire elles disposent de telles données mais ne permettent pas de calcul, comme Wheelmap.

Nous avons ensuite fait une large revue des différents algorithmes connus pour le calcul d'itinéraire. Nous avons en particulier cité Dijkstra, le plus connu des algorithmes de recherche de routes. Plusieurs algorithmes sont en fait des dérivés de Dijkstra qui usent d'une heuristique pour améliorer le rapport entre la qualité de l'itinéraire obtenu et l'efficacité du calcul. C'est le cas en particulier de A* qui ne visite pas tous les sommets et qui oriente l'exploration vers la destination finale. C'est le cas aussi des contractions hiérarchiques, qui contractent le graphe en ôtant des nœuds intermédiaires selon des règles assez complexes. Nous sommes arrivés à la conclusion que des systèmes largement utilisés de calcul d'itinéraires routiers ou piétonniers, comme Google ou ViaMichelin se servent de dérivés de l'algorithme de Dijkstra qui usent d'une heuristique sophistiquée faisant partie de leurs secrets de fabrication. Les systèmes *open source*, quant à eux, indiquent souvent l'utilisation de Dijkstra, de A* ou des contractions hiérarchiques.

Nous avons passé ensuite en revue des sources de cartes existantes : Google Maps API, ViaMichelin, OpenStreetMap, ArgGIS et Swisstopo. Nous étions en effet dépendant des cartes que nous ne pouvons pas obtenir par nous-mêmes. La seule solution qui permît de récupérer le graphe que forme le réseau de routes et de rue était OpenStreetMap, qui était *open source*. Cette solution nous aurait permis de faire les calculs nous-mêmes. Les autres systèmes permettaient au mieux de calculer gratuitement un itinéraire, au pire de le calculer moyennant finance. Nous avons cependant dû constater que les données d'OpenStreetMap, dépendantes des contributions des utilisateurs, n'étaient pas partout complètes.

Dans la **deuxième partie** du travail, nous sommes passé au développement proprement dit du calcul d'itinéraire. Nous avons retenu Google Maps JavaScript API pour deux raisons. En premier lieu parce que l'application que nous avons reçue utilisait déjà Google Maps et ensuite parce que cette solution permettait un calcul de plusieurs itinéraires, gratuit jusqu'à un certain point, c'est-à-dire jusqu'à 2'500 requêtes quotidiennes et 50 requêtes par seconde.

Nous avons donc exploité le service Directions de cette API pour calculer l'itinéraire d'un point à un autre. Quand on lui passe un objet DirectionsRequest, il retourne un objet DirectionsResult. Cet objet assez complexe contient entre autres la ou les routes à parcourir. Nous avons constaté qu'il retourne deux ou trois itinéraires si on lui demande de fournir des routes alternatives. Dans chaque route, il y a un tableau d'objets LatLng, nommé overview_path. Grâce à ce tableau, nous avons pu créer la Polyline qui représente le tracé du chemin sur la carte. C'était la première étape du développement.

Dans une deuxième étape, nous devions afficher uniquement les points qui se trouvent sur les différentes routes calculées, en affichant en rouge les marqueurs qui remplissaient au moins un des critères négatifs cochés par l'utilisateur et en vert les autres, mais en masquant tous les points éloignés de la route.

Pour cela, la librairie Geometry de Google Maps JavaScript API nous a permis au moyen de la méthode isLocationOnEdge(point, polyline, tolerance) de déterminer si chaque point d'intérêt sur la carte se trouve sur le trajet, avec la tolérance exprimée en degrés de longitude ou de latitude. Nous avons pu déterminer qu'il faut considérer qu'un degré de latitude mesure environ 111 km et qu'un degré de longitude vaut environ 76.5 km sous nos latitudes. Nous avons donc estimé que si la tolérance est de 10^{-3}, cela représente une distance de l'ordre de l'hectomètre. Nous avons choisi une tolérance de 0.0005, donc environ 50 mètres.

Afin de rendre l'application plus efficace et d'éviter de confronter chaque point à chacune des trois routes, nous avons d'abord sélectionné les points qui se trouvent dans une ellipse dont l'origine et la destination du trajet sont les foyers. Nous avons choisi comme grand axe de l'ellipse le double de la distance entre les foyers. Cependant, le grand axe ne peut être inférieur à cette distance additionnée du double de la tolérance. Nous avons pu tester de manière convaincante que cette présélection des points au moyen de l'ellipse augmentait l'efficacité de la fonction qui passait en revue tous les points d'un facteur compris entre 5.4 et 6.6.

Nous avons ensuite traité les points qui restaient dans l'ellipse afin qu'ils soient de la bonne couleur, rouge s'ils avaient au moins un obstacle coché par l'utilisateur, pente trop grande, trottoir trop étroit ou obstacle.

Dans la troisième étape du développement, nous avons aussi permis à l'utilisateur de sélectionner l'une ou l'autre route et affiché la longueur de cette route. Nous n'avons pas affiché le temps car il est très différent en fonction de la personne à mobilité réduite et de son matériel. Nous avons fait d'autres aménagements cosmétiques : ajout de marqueurs au début et à la fin de la route, recentrage de la carte de manière à ce qu'elle englobe tous les trajets.

Une quatrième étape s'est éloignée de notre tâche de calcul d'itinéraire. Nous avons, en accord avec la direction du projet, apporté quelques modifications aux fenêtres (InfoWindow) qui s'ouvrent quand on clique sur un des marqueurs de la carte. Alors que l'application comportait initialement deux colonnes avec à gauche le texte et à droite la photo, nous avons fait une seule colonne avec la photo en tête de colonne. En effet, dans la version reçue, la colonne de texte était trop étroite sur un téléphone. Nous avons aussi affiché tous les critères d'accessibilité en grisant ceux qui sont absents, plutôt que de n'afficher que les présents.

La **troisième partie** de notre travail a consisté à établir un état de l'art de la motivation au *crowdsourcing*. Ce terme, dérivé de *outsourcing* et *crowd* représente l'externalisation d'un problème auprès de la foule.

Nous avons vu qu'il existe plusieurs types de *crowdsourcing*. Dans le cas de WE-MAP, il est demandé aux gens d'effectuer une tâche simple, mais à grande échelle. Cela ne demande pas de capacités cognitives ni créatives particulières. Nous avons vu que la motivation peut être intrinsèque, c'est-à-dire liée directement à l'accomplissement de la tâche, ou au contraire extrinsèque. À cela s'ajoute la participation involontaire comme dans le cas de ReCaptcha, mais nous n'avons pas trouvé comment appliquer cette solution à notre application.

La motivation intrinsèque à contribuer à WE-MAP peut résulter de l'aspect social de l'application. Vouloir aider les personnes handicapées peut être un moteur pour bien des gens. De plus, les personnes concernées, quand elles ont exploré un lieu, peuvent vouloir contribuer pour le bien de leurs pairs. La ludification entre aussi en considération, avec par exemple un classement des meilleurs contributeurs par zone géographique. Ce classement devrait être limité dans le temps, afin d'éviter de décourager ceux qui sont tout en bas ainsi que les entrants.

La motivation extrinsèque peut être une rémunération, mais il faut trouver les sources de financement. Nous avons évoqué des bons chez des sponsors. Une autre approche serait de limiter l'usage de l'application et de l'élargir pour les grands contributeurs. Cependant nous avons trouvé deux défauts à cette solution : d'abord elle ne motive pas les valides et ensuite ceux qui ont le plus besoin de l'application sont ceux qui auront le plus de peine à récolter des informations.

Dans tous les cas, il faut rester vigilant car si la foule est motivée pour les mauvaises raisons, elle pourrait contribuer à tout prix pour bénéficier des récompenses, en privilégiant la quantité plutôt que la qualité des données.

La **quatrième partie** de notre application consistait à faire une analyse qualitative de l'application. Nous avons choisi la méthode des *focus groups* ou groupes de discussion. Nous n'avions pas le temps pour une large étude, mais nous avons formé deux groupes d'utilisateurs qui ont testé l'application durant une semaine, aidés d'instructions.

Globalement, les participants ont jugé l'application utile et certains ont même regretté de ne pas en avoir disposé pour leurs grand-parents. Les participants n'étaient, par contre, pas pleinement convaincus de la possibilité de motiver la foule. Un participant a proposé d'impliquer le personnel des offices de tourisme durant les périodes creuses, voire les organisateurs de manifestations.

Les testeurs ont aussi fait un certain nombre de remarques sur l'application en tant que telle. Certaines de ces remarques nous ont amenés à faire des modifications dans le cadre de ce travail. Pour d'autres, nous n'avons pas effectué de changement car les choix qui avaient été faits résultaient des expériences des employés de la poste qui ont saisi des centaines de points.

Ils ont estimé que sur la page de saisie des critères, la carte est trop petite. Nous avons maintenu cet état de fait, car il s'agit de laisser un maximum de place pour les POI listés dans le menu « Ajouter un endroit existant », afin qu'il en apparaisse plus de deux sur un téléphone disposant d'un petit écran.

Sur la carte, les utilisateurs ont eu de la peine à trouver le bouton de calcul d'itinéraire. Ils ont proposé de changer l'icône afin qu'elle ressemble à celle de Google, bien connue, ce que nous avons fait. Comme l'icône direction de Google ne fait pas partie des icônes du *framework* JQuery mobile, il a fallu créer une icône personnalisée.

Dans la fenêtre d'information qui s'ouvre lorsque l'on clique sur un marqueur, les participants n'ont pas trouvé nette la différence entre les critères présents, en noir, et absents, grisés. Nous avions apporté cette modification dans la quatrième étape du développement. Certains ont proposé d'ôter les critères absents, d'autres de les barrer, d'autres enfin de les mettre bien plus clairs. C'est cette dernière solution que nous avons choisie, faisant passer l'opacité de 40 % à 20 %. Sur la suggestion d'un des participants, nous avons aussi intervertis les boutons « Supprimer » et « Modifier » dans cette même fenêtre.

Pour que la signification des icônes représentant les critères soit plus claire, nous avons aussi ajouté des *tooltips* qui affichent la légende de l'icône quand on passe le pointeur de la souris dessus sur un ordinateur ou quand on clique dessus pour les écrans tactiles.

Nous avons aussi ajouté un message d'erreur quand la géolocalisation ne fonctionne pas, par exemple quand elle n'est pas activée dans le navigateur ou sur le téléphone utilisé. En effet, il ne se passait rien quand l'utilisateur cliquait sur « Ma position » et il pouvait penser que c'était un bug de l'application.

Nous avons enfin créé un tutoriel de cinq écrans, actuellement en français et anglais mais facilement *localisable* en d'autres langues, afin de faciliter le travail. On accède à ces tutoriels depuis le premier écran de bienvenue qui permet aussi de choisir la langue.

L'Annexe VI résume l'entier des modifications que nous avons apportées à l'application durant ce travail de Bachelor.

*

Nous pouvons formuler **quelques recommandations** quant au développement de l'application.

Si elle devait avoir le succès que nous lui souhaitons, l'application pourrait se voir limitée dans son utilisation par le nombre de requêtes autorisés pour le service Directions de Google Maps JavaScript API, qui est de de 50 par secondes et surtout de 2'500 par 24 heures. Il faudrait alors soit payer pour augmenter ce dernier nombre, ce qui est possible jusqu'à 100'000, moyennant UDS 0.50 pour 1'000 requêtes, soit trouver une manière de rationner les interrogations, soit changer complètement de technologie.

Nous avons fixé la tolérance à 0.0005°. Nous avons vu que cela représentait entre 37.9 et 55.5 m. Cette valeur devrait peut-être réévaluée en fonction des réels besoins. Si elle est trop petite, il n'apparaîtra pas assez de points et si elle est trop grande, il en apparaîtra trop et l'utilisateur ne s'y retrouvera pas dans une forêt de marqueurs.

Pour l'introduction des informations, il vaudrait la peine d'intéresser les offices de tourisme à saisir celles qui les concernent. Cela apporterait une valeur ajoutée à leur région et à notre application.

Pour la motivation des contributeurs, une ludification avec système de classement par région et par période pourrait être une bonne idée. Il faut en tout cas être prudent avec les systèmes qui incitent les gens à collaborer pour de mauvaises raisons, car ils pourraient être incités à mettre des données de mauvaise qualité en privilégiant la quantité. Dans tous les

cas, une méthode permettant de réguler le système pour éliminer les fausses informations doit être envisagée.

*

Dans sa partie du calcul des itinéraires, ce travail s'est limité à détecter et traiter les points qui se trouvaient sur les trajets. Dans une version ultérieure, on pourrait prévoir de proposer à l'utilisateur de calculer un itinéraire alternatif si l'obstacle signalé lui semble insurmontable.

Pour le *crowdsourcing*, c'est seulement quand l'application sera disponible pour les utilisateurs finaux que nous saurons si les gens sont motivés à contribuer. Le *crowdsourcing* fait la grande force de notre application, car il permet à l'infini d'améliorer ses données, mais c'est aussi sa faiblesse, car si personne ne contribue, elle ne vaut rien. Plus il y aura d'informations dans l'application et plus elle sera attractive. Plus elle sera attractive, plus il y aura d'utilisateurs et donc plus ils pourront saisir des informations. La difficulté est d'amorcer le processus. Mais les responsables du projet en sont vraisemblablement conscients.

Quant à notre évaluation par les *focus groups*, sa portée est limitée par le petit nombre de participants et par le fait qu'ils n'étaient pas en situation de handicap. Elle nous a cependant permis certaines améliorations. Elle est ainsi mieux préparée pour des évaluations ultérieures à plus grande échelle, avec les personnes concernées.

Cette application a un grand potentiel si Google ou un autre géant de l'informatique ne décide pas de lui donner une petite sœur, car, évidemment, leurs moyens sont sans commune mesure avec les nôtres.

Références

Alsedà, L. (2017, janvier 26). *AStar-Algorithm.pdf.* Récupéré sur Universitat Autònoma de Barcelona - UAB Barcelona: http://mat.uab.cat/~alseda/MasterOpt/AStar-Algorithm.pdf

Andro, M., & Saleh, I. (2015, Juin 16). *La Correction participative de l'OCR*. Récupéré sur Bulletin des Bibliothèques de France: http://bbf.enssib.fr/contributions/la-correction-participative-de-l-ocr

Bast, H. (2012a, Juin 6). *Efficient Route Planning, SS 2012, Lecture 6 [Microsoft PowerPoint].* Récupéré sur Site web de l'Université de Fribourg (Allemagne): http://ad-teaching.informatik.uni-freiburg.de/route-planning-ss2012/lecture-6.pdf

Bast, H. (2012b, Juin 13). *Efficient Route Planning, SS 2012, Lecture 7 [Microsoft PowerPoint].* Récupéré sur Site de l'Université de Fribourg (Allemagne): http://ad-teaching.informatik.uni-freiburg.de/route-planning-ss2012/lecture-7.pdf

Bonifas, F. (2013, Septembre). *MoodWalkR : conception et réalisation d'une application de calcul d'itinéraires adaptés aux piétons [Rapport de stage].* Récupéré sur Site du master géomatique Sigma de l'Université de Toulouse: http://sigma.univ-toulouse.fr/_resources/th%25C3%25A9matiques/pdf/Stages2013/Bonifas2013_Rapport.pdf?download=true

Brown, M. (2007, avril 26). *Londonist Interviews... OpenStreetMap Guru Steve Coast*. Récupéré sur Londonist: http://londonist.com/2007/04/londonist_inter_11

Byrne, M. (2015, Mars 22). *The Simple, Elegant Algorithm That Makes Google Maps Possible*. Récupéré sur Motherboard: https://motherboard.vice.com/en_us/article/the-simple-elegant-algorithm-that-makes-google-maps-possible

CartoType. (2016). *Offline routing for large areas*. Récupéré sur Cartotype: http://www.cartotype.com/articles/25-offline-routing-for-large-areas

Champeau, G. (2012, Mars 30). *Comment Google exploite le spam pour améliorer Google Maps*. Récupéré sur Numerama: http://www.numerama.com/magazine/22201-comment-google-exploite-le-spam-pour-ameliorer-google-maps.html

Coline B. (2016, Septembre 16). *Crowdsourcing gamifié : exploitation déguisée ou solution gagnant-gagnat ?* Récupéré sur La Gamification: http://www.lagamification.com/crowdsourcing-gamifie-exploitation-deguisee-solution-gagnant-gagnant/

Commission générale de terminologie et de néologie. (2014). Production participative. *Journal Officiel de la République Française*(0179), 12995. Récupéré sur https://www.legifrance.gouv.fr/affichTexte.do;jsessionid=?cidTexte=JORFTEXT000029331922&dateTexte=&oldAction=dernierJO&categorieLien=id

Coutinho, A., Neto, F. R., & Perna, C. E. (2013, Septembre). Determination of normative values for 20 min exercise of wheelchair propulsion by spinal cord injury patients. *Spinal Cord, 51*(10), 755-760. doi:10.1038/sc.2013.89

Delamarck, S., & Pardanaud, J. (2017, 04 13). *Qu'est-ce que l'Ajax ? - Dynamisez vos sites Web avec JavaScript !* Récupéré sur Openclassrooms: https://openclassrooms.com/courses/dynamisez-vos-sites-web-avec-javascript/l-ajax-qu-est-ce-que-c-est

Digital Communications Division in the U.S. Department of Health and Human Services' (HHS) Office of the Assistant Secretary for Public Affairs. (s.d.). *Focus Groups*. Récupéré sur Usability.gov: https://www.usability.gov/how-to-and-tools/methods/focus-groups.html

Dijkstra, E. W. (1971). *A short introduction to the art of programming (EWD 316).* Eindhoven: Techn. Hogeschool.

ESRI. (s.d.). *ArcGIS Pro*. Consulté le avril 20, 2017, sur Esri: GIS Mapping Software, Spatial Data Analytics & Location Platform: http://www.esri.com/en/arcgis/products/arcgis-pro/overview

Geisberger, R. (2008, Juillet 1). *Contraction Hierarchies: Faster and Simpler Hierarchical*

Routing in Road Networks (Thèse de diplôme). Récupéré sur Site de l'Institut d'informatique théorique de l'Université de Karlsruhe: http://algo2.iti.kit.edu/documents/routeplanning/geisberger_dipl.pdf

Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008). Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. *7th Workshop on Experimental Algorithms (WEA)* (pp. 319-333). Berlin: Springer. Récupéré sur http://algo2.iti.kit.edu/schultes/hwy/contract.pdf

Geographic Information Technology Training Alliance (GITTA). (2016, 04 28). *Dijkstra Algorithm: Short terms and Pseudocode*. Récupéré sur GITTA Geographic Information Technology Training Alliance: http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html

Gervais, B., & Morin, G. (s.d.). *Handimap.org - la ville accessibles à tous*. Consulté le Avril 19, 2017, sur Handimap.org: http://www.handimap.org/

Golshan, B. (2015, Octobre 21). *What algorithm is used by Google Maps?* Récupéré sur Quora: https://www.quora.com/What-algorithm-is-used-by-Google-Maps

Google. (2017a, avril 6). *Guide du développeur | Google Maps Directions API | Google Developpers*. Récupéré sur Google Developpers: https://developers.google.com/maps/documentation/directions/intro?hl=fr

Google. (2017b, avril 6). *Guide du développeur | Google Maps Distance Matrix API | Google Developpers*. Récupéré sur Google Developpers: https://developers.google.com/maps/documentation/directions/intro?hl=fr

Google. (2017c, avril 6). *Limites d'utilisation de Google Maps Directions API | Google Maps Directions API | Google Developpers*. Récupéré sur Google Developpers: https://developers.google.com/maps/documentation/directions/usage-limits?hl=fr

Google. (2017d, Avril 6). *Sélecteur d'API | Google Maps API | Google Developpers*. Récupéré sur Google Developpers: https://developers.google.com/maps/documentation/api-picker?hl=fr

Google. (2017e, Avril 6). *Service Directions | Google Maps Javascript API*. Récupéré sur Google Developpers: https://developers.google.com/maps/documentation/javascript/directions

Google. (2017f, Avril 6). *Bibliotheque Geometry | Google Maps Javascript API*. Récupéré sur Google Developpers: https://developers.google.com/maps/documentation/javascript/geometry

Google. (2017g, Avril 6). *Google Maps JavaScript API V3 Reference | Google Maps Javascript API*. Récupéré sur Google Developpers: https://developers.google.com/maps/documentation/javascript/reference

Grier, D. (2013). *Crowdsourcing for Dummies.* Chichester: John Wiley & Sons.

Haklay, M., & Weber, P. (2008). OpenStreetMap: User-Generated Street Map [PDF]. *IEEE Pervasive Computing*, 12-18. Récupéré sur http://discovery.ucl.ac.uk/13849

Hossain, M. (2012). Crowdsourcing: Activities, incentives and users' motivations to participate. *International Conference on Innovation, Management and Technology Research* (pp. 501-506). Malacca: IEEE. doi:10.1109/ICIMTR.2012.6236447

Howe , J. (2009). *How the Power of the Crowd is Driving the Future of Business.* London: Random House Business Books.

Infomaniak. (s.d.). *Rediriger tous les visiteurs sur le site avec https (SSL)*. Consulté le 07 01, 2017, sur Base de connaissances Infomaniak: https://www.infomaniak.com/fr/support/faq/1961/rediriger-tous-les-visiteurs-sur-le-site-avec-https-ssl

Jungnickel, D. (2008). *Graphs, Networks and Algorithms.* Berlin: Springer.

Kamranzafar. (2012, Août 23). *JQuery Mobile Android-style Toast popup* . Récupéré sur GitHubGist: https://gist.github.com/kamranzafar/3136584

karussell. (2017, Mars 24). *GraphHopper. Technical Overview*. Récupéré sur GitHub: https://github.com/graphhopper/graphhopper/#technical-overview

Lacomme, P., Prins, C., & Sevaux, M. (2007). *Algorithmes de graphes.* Paris: Eyrolles.

Layas, F., & Petrie, H. (2016). Exploring Intrinsic and Extrinsic Motivations to Participate in a

Crowdsourcing Project to Support Blind and Partially Sighted Students. Dans H. Petrie, J. Darzentas, T. Walsh, D. Swallow, L. Sandoval, A. Lewis, & C. Power, *Universal Design 2016: Learning from the Past, Designing for the Future* (Vol. 229, pp. 545-555). Amsterdam: IOS Press Book. doi:10.3233/978-1-61499-684-2-545

Lebraty, J.-F., & Lobre, K. (2015). *Crowdsourcing: porté par la foule.* Londres: ISTE Editions.

lessonsharing, teoli, Robitaille, J., BenoitL, Mgjbot, Chbok, & Laurent, D. (2017, Mars 24). *XMLHttpRequest - Référence Web API*. Récupéré sur Mozilla Developper Network: https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest

Liu, Z., Glassey Balet, N., Sokhn, M., & De Gaspari, E. (2017). Crowdsourcing-Based Mobile Application for Wheelchair Accessibility. *Journal on Technology & Persons with Disabilities, 5*, 1-15.

Michelucci, P., & Dickinson, J. L. (2016, Janvier 1). The power of crowds. *Science, 351*(6268), 32-33. doi: 10.1126/science.aad6499

Ministère de l'Éducation nationale (français). (s.d.). *L'ellipse*. Récupéré sur Eduscol: http://eduscol.education.fr/orbito/orb/meca/meca112.htm

Morschheuser, B., Hamari, J., & Koivisto, J. (2016). Gamification in Crowdsourcing: A Review. *49th Annual Hawaii International Conference on System Sciences (HICSS).* Hawaii: IEEE. doi:10.1109/HICSS.2016.543

Mozilla Foundation. (s.d.). *Geolocation.getCurrentPosition()*. Consulté le Juillet 15, 2017, sur MDN Mozilla Developer Network: https://developer.mozilla.org/fr/docs/Web/API/Geolocation/getCurrentPosition

Nosrati, M., Karimi, R., & Hasanvand, H. (2012). Investigation of the * (Star) Search Algorithms: Characteristics, Methods and Approaches. *World Applied Programming, 2*(4), 251-256.

Opendata swiss. (s.d.). *Office fédéral de topographie Swisstopo*. Consulté le Avril 23, 2017, sur Opendata.swiss: https://opendata.swiss/fr/organization/bundesamt-fur-landestopografie-swisstopo

OpenStreetMap. (2017a, Mars 15). *Barrier*. Récupéré sur OpenStreetMap Wiki: http://wiki.openstreetmap.org/wiki/Key:barrier

OpenStreetMap. (2017b, mars 18). *Databases and data access APIs*. Récupéré sur OpenStreetMap Wiki: http://wiki.openstreetmap.org/wiki/Databases_and_data_access_APIs

OpenStreetMap. (2017c, Avril 22). *Frameworks*. Récupéré sur OpenStreetMap Wiki: http://wiki.openstreetmap.org/wiki/Frameworks#Navigation

OpenStreetMap. (2017d, avril 17). *Overpass API*. Récupéré sur OpenStreetMap Wiki: http://wiki.openstreetmap.org/wiki/Overpass_API

OpenStreetMap. (2017e, Avril 3). *PostgreSQL*. Récupéré sur OpenStreetMap Wiki: http://wiki.openstreetmap.org/wiki/PostgreSQL

Patel, A. (2017, Mars 16). *Implementation notes*. Récupéré sur Stanford University - Stanford Theory Group - Amit's A* Pages: http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html

pgRouting Community. (s.d.). *pgRouting Library*. Consulté le Avril 29, 2017, sur Site de pgRouting: http://pgrouting.org/index.html

Prins, C. (1996). Comparaison d'algorithmes de plus courts chemins sur des graphes routiers de grande taille. *RAIRO - Operations Research - Recherche Opérationnelle 30.4*, pp. 333-357.

Rosario, K. (2011, Décembre 17). *Why ReCaptcha works: The 4 requirements of Crowdsourcing*. Consulté le Juillet 02, 2017, sur KeithRosario.com: https://www.keithrozario.com/2011/12/recaptcha-4-requirements-crowdsourcing.html

Same Origin Policy. (2010, Janvier 6). *Same Origin Policy*. Récupéré sur World Wide Web Consortium - Web Security: https://www.w3.org/Security/wiki/Same_Origin_Policy

Sanders, P., & Schultes, D. (2007). Engineering Fast Route Planning Algorithms. *WEA 2007, LNCS 4525* (pp. 23-36). Rome: Springer-Verlag Berlin Heidelberg.

Schenk, E., & Guittard, C. (2012). Une typologie des pratiques de Crowdsourcing : l'externalisation vers la foule, au-delà du processus d'innovation. *Management*

International, 16, 89-100. doi:10.7202/1012395ar
Sedgewick, R., & Vitter, J. (1984). Shortest paths in Euclidean Graphs (extended abstract). *Foundations of Computer Science, 25th Annual Symposium on* (pp. 417-424). Providence: Department of Computer Science, Brown University. doi:10.1109/SFCS.1984.715943
Segal, Z., & Duron, Y. (2015). *La motivation, une compétence qui se développe: guide pour développer la motivation et l'engagement au travail.* Montreuil: Pearson.
Smith, D., Manesh, M., & Alshaikh, A. (2013). How Can Entrepreneurs Motivate Crowdsourcing Participants? *Technology Innovation Management Review.*, 23-30. Récupéré sur https://timreview.ca/sites/default/files/article_PDF/Smith_et_al_TIMReview_February2013.pdf
Sobczak, S., & Groß, M. (2010). *Crowdsourcing: Grundlagen und Bedeutung für das E-Business.* Boizenburg: Hülsbusch.
Sonenblum, S. E., Sprigle, S., & Lopez, R. A. (2012). Manual Wheelchair Use: Bouts of Mobility in Everyday Life. *Rehabilitation Research and Practice*, ID 753165. doi:10.1155/2012/753165
Swiss OpenStreetMap Association [SOSM]. (2012, juin 20). *Statuts*. Récupéré sur Swiss OpenStreetMap Association: http://sosm.ch/fr/about-2/statuts/
Swisstopo. (s.d.). Consulté le Avril 24, 2017, sur Site web GeoAdmin API: https://api3.geo.admin.ch/index.html
Tacchini, C. (2016, Septembre 29). Bachelor's Thesis 2017. Sierre, Valais, Suisse. Récupéré sur http://intranet.hevs.ch/Portals/31/Formations/IG/registred/TB/TB_BA_presentation%202017_EN.pdf?ver=2016-09-29-113136-637
Teulings, T. (2012, Décembre 12). *Mailing Lists*. Récupéré sur libosmscout: https://sourceforge.net/p/libosmscout/mailman/message/28823196/
Teulings, T. (2016, Mai 29). *Features. Routing*. Récupéré sur libosmscout: http://libosmscout.sourceforge.net/features/routing/
Tolerico, M. L., Ding, D., Cooper, R. A., Spaeth, D. M., Fitzgerald, S. G., Cooper, R., . . . Boninger, M. L. (2007). Assessing mobility characteristics and activity levels of manual wheelchair users. *Journal of Rehabilitation Research & Development, 44*(4), 561-572. Récupéré sur http://www.rehab.research.va.gov/jour/07/44/4/Tolerico.html
Universität Heidelberg. (2016). *OpenRouteService APIs*. Récupéré sur OpenRouteService API Documentation: http://openrouteservice.readthedocs.io/en/1.0/index.html
von Ahn, L. (2011, Avril). Massive-scale online collaboration. *TED Talk*. Consulté le Juillet 2, 2017, sur https://www.ted.com/talks/luis_von_ahn_massive_scale_online_collaboration
w3schools. (s.d.). *How to Create Tooltips*. Consulté le Juillet 12, 2017, sur w3schools: https://www.w3schools.com/howto/howto_css_tooltip.asp
Wales, J. (2013, Avril 4). *Why does Jimmy Wales think crowdsourcing is a horrible term?* Récupéré sur Quora: https://www.quora.com/Wikipedia/Why-does-Jimmy-Wales-think-crowdsourcing-is-a-horrible-term#
wheelmap.org. (s.d.). *FAQ - news von Wheelmap.org*. Consulté le avril 20, 2017, sur Wheelmap.org: https://news.wheelmap.org/faq/

Annexe I : cahier des charges

Hes·so VALAIS WALLIS
Haute Ecole de Gestion & Tourisme
Hochschule für Wirtschaft & Tourismus

Cahier des charges
Travail de Bachelor 2017

WE-MAP

Crowdsourced map based mobile application for the wheelchair accessibility

Professeur Nicole Glassey Balet

Étudiant David Mack

David Mack
15/03/2017

Table des matières

1. Introduction

Ce travail de bachelor s'inscrit dans le projet WE-MAP de l'Institut d'informatique de gestion de la Haute École Spécialisée de Suisse Occidentale (HES-SO) Valais-Wallis.

2. Contexte du travail de diplôme

L'accessibilité des lieux publics est un problème essentiel pour les personnes à mobilité réduite. De nombreuses applications permettent de calculer un itinéraire porte-à-porte à pied, à vélo ou en voiture, voire en transports publics. Cependant, aucune ne tient compte des obstacles que rencontrerait une personne à mobilité réduite sur le parcours.

Le projet WE-MAP de l'institut d'informatique de gestion de la HES-SO Valais-Wallis vise à fournir une plateforme de génération d'itinéraires pour les personnes à mobilité réduite, avant tout les personnes en fauteuil roulant, mais aussi les personnes âgées ou les familles avec poussette. Une application HTML5/PHP/JS/MySQL a commencé à être développée à l'institut. Elle permet pour l'instant de saisir des points d'intérêt, en anglais *points of interest* (POI), en précisant les facteurs d'accessibilités à ces endroits, mais elle ne gère pas du tout le calcul d'itinéraire. Environ 800 POI y ont déjà été saisis, grâce au concours des facteurs de Sion, de Sierre et du Val d'Anniviers.

3. Problématique

L'un des intérêts de l'application réside dans le fait qu'elle permettra aux personnes en situation de mobilité réduite de calculer un chemin qui évite les obstacles : pentes trop fortes, trottoirs trop étroits, etc… Elle devra donc calculer le chemin d'un point A à un point B, en évitant les obstacles. Il ne s'agira pas de réinventer la roue, mais d'étudier les algorithmes existant dans la littérature, d'en retenir un pour l'appliquer au cas qui nous intéresse et enfin de l'introduire dans l'application Web existante.

L'application ne pourra être efficace que si elle dispose de données exhaustives sur les obstacles existants. Une des difficultés réside donc dans le recensement de ces obstacles. Pour cela, nous comptons sur l'engagement des utilisateurs qui doivent saisir ces informations. C'est ce que l'on appelle le *crowdsourcing* : la « foule » des usagers va participer à la récolte d'information. Cela pose le problème de la motivation de ces usagers, qui doivent se mobiliser pour glaner un maximum de renseignements.

Il faudra enfin contrôler la qualité des informations et effectuer des tests avec les données collectées, afin de s'assurer que l'application sera utile à ceux à qui elle est destinée.

4. Travail à effectuer

Les buts de ce travail de diplôme sont de

- établir l'état de l'art concernant l'algorithme de génération automatique d'itinéraires porte-à-porte et choisir la meilleure solution à intégrer dans la future application
- développer l'application web existante à partir du choix de l'algorithme de génération d'itinéraires fait auparavant.
- comparer et étudier les méthodologies de *crowdsourcing* existantes, en particulier en ce qui concerne la motivation des contributeurs, la sécurité et la sauvegarde des données
- Analyser l'application de la manière qui semblera la plus appropriée

5. Planification

S'agissant d'un travail à réaliser par un seul étudiant, nous renonçons à utiliser la méthode Scrum, certaines des cérémonies, comme par exemple le daily meeting, perdant tout sens. De plus, il ne sera pas possible de négocier à chaque sprint les user stories devant être développées, car d'une part le travail ne comporte pas qu'une partie de développement et d'autre part le cahier des charges étant fixé au début, il n'y a que peu de négociations possibles. Enfin, il n'y aura pas de client à qui présenter les résultats.

Il semble cependant opportun de conserver de la méthodologie agile le travail par itérations, avec rencontres des responsables du projet tous les quinze jours et analyse du travail effectué, afin de s'assurer de la correspondance du travail avec les attentes.

Ce travail de Bachelor s'effectue au début en parallèle avec les cours, dont le nombre d'heures sera réduit dès fin avril et qui se terminent avec les examens en juin. Selon le planning de l'école, 11 heures hebdomadaires sont consacrées à ce travail du 20 février au 30 avril, avec une semaine de vacances à Pâques, puis 15 heures hebdomadaires jusqu'au 18 juin et ensuite 24 heures hebdomadaires, sauf la semaine d'examens du 26 juin au 2 juillet.

5.1. Phase 0

Cette phase sera dédiée à la planification et à la prise en main du projet. Elle se terminera avec la remise du cahier des charges le 15 mars 2017.

5.2. Phase 1 : calcul d'itinéraires porte-à-porte

Cette phase portera sur les algorithmes et techniques de calcul des itinéraires porte-à-porte en tenant compte des obstacles potentiels. Elle devrait s'achever le 13 avril 2017.

5.3. Phase 2 : développement

Durant cette phase, le résultat de l'étude précédente va être intégré dans l'application existante. Il s'agit du cœur du travail de Bachelor, il devra se terminer vers le 15 juin 2017.

5.4. Phase 3 : *crowdsourcing*

Cette phase sera consacrée à la recherche sur l'approche du *crowdsourcing*, en particulier ses aspects de motivation et de sauvegarde des données et finira le 6 juillet 2017.

5.5. Phase 4 : évaluation

Cette phase consistera à évaluer l'application obtenue. La méthode d'évaluation retenue sera définie plus tard durant le travail. Elle sera faite pour le 20 juillet 2017.

5.6. Rédaction

Le travail de Bachelor doit être rendu le 9 août à midi.

6. Échéancier

Début du projet :	20.02.2017
Remise du cahier des charges :	15.03.2017
État de l'art sur le calcul d'itinéraires :	13.04.2017
Intégration du calcul d'itinéraires :	15.06.2017
Recherche sur le *crowdsourcing* :	06.07.2017
Évaluation :	20.07.2017
Remise du travail de bachelor :	09.08.2017

7. Délivrables

7.1. Rapport d'analyse

Le rapport d'analyse contiendra l'état de l'art sur la génération automatique d'itinéraires, la description des éléments apportés à l'application et le résultat des recherches sur le *crowdsourcing*. Il contiendra aussi les éléments de contrôle de la qualité de l'information saisie et le résultat des tests avec les données collectées.

7.2. Application

L'application devra contenir le calcul d'itinéraires en évitant les obstacles.

8. Conclusion

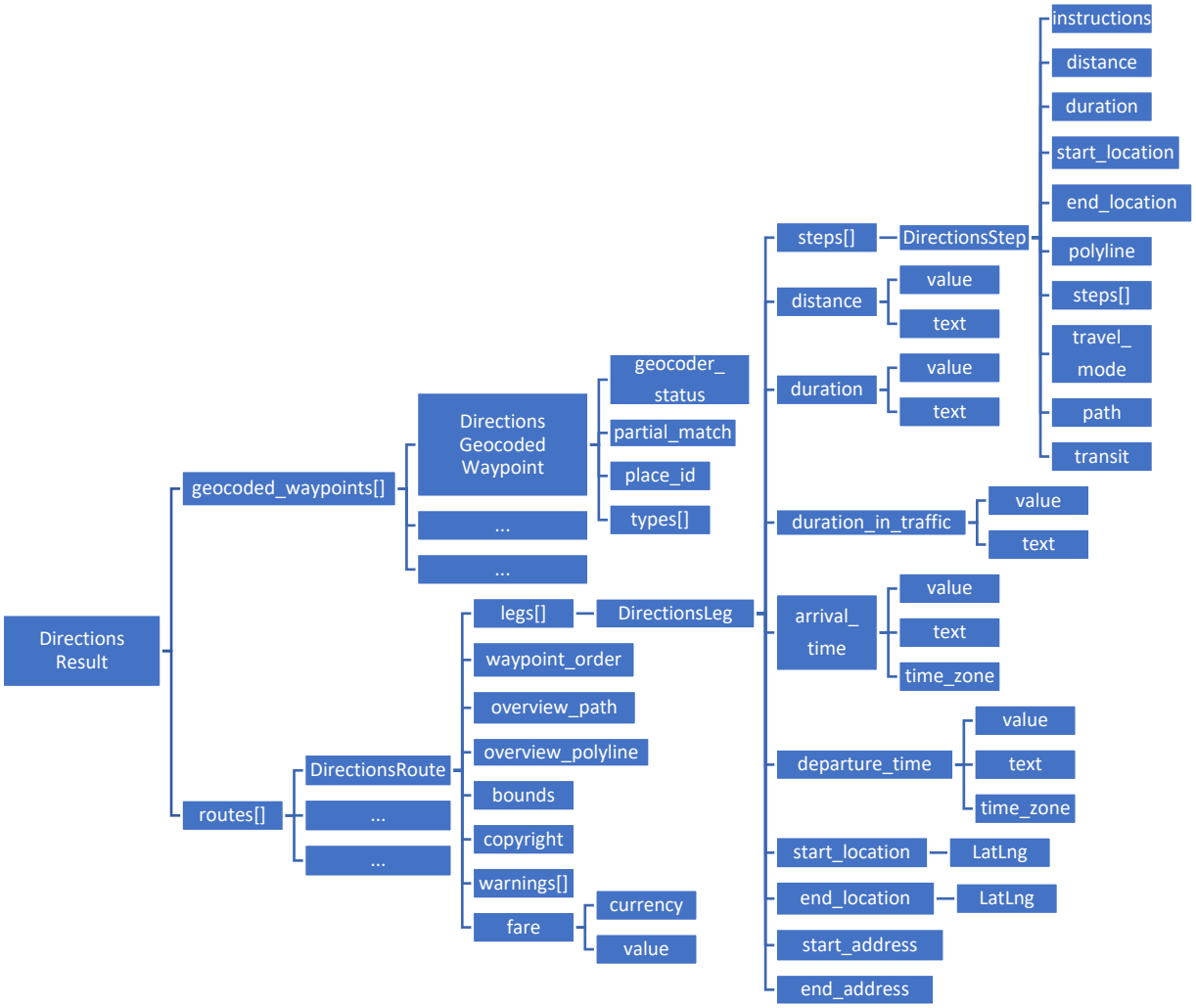
En début de projet, alors que d'une part l'on ne sait pas encore ce qu'il ressortira de la recherche sur l'état de l'art et que d'autre part l'application existante n'a été que partiellement découverte, il est difficile de faire une planification plus précise pour le développement de la dite-application. Il nous semble cependant que l'important est posé puisque les étapes clefs ont été définies avec leurs délais de réalisation.

Annexe I : Planning

Semaine	Lundi	Dimanche	Heures	Phase
Semaine 1	20.02.2017	26.02.2017	11	Planification
Semaine 2	27.02.2017	05.03.2017	11	Planification
Semaine 3	06.03.2017	12.03.2017	11	Planification
Semaine 4	13.03.2017	19.03.2017	11	État de l'art itinéraires
Semaine 5	20.03.2017	26.03.2017	11	État de l'art itinéraires
Semaine 6	27.03.2017	02.04.2017	11	État de l'art itinéraires
Semaine 7	03.04.2017	09.04.2017	11	État de l'art itinéraires
Semaine 8	10.04.2017	16.04.2017	11	État de l'art itinéraires
Vacances	17.04.2017	23.04.2017	0	
Semaine 9	24.04.2017	30.04.2017	11	Développement
Semaine 10	01.05.2017	07.05.2017	15	Développement
Semaine 11	08.05.2017	14.05.2017	15	Développement
Semaine 12	15.05.2017	21.05.2017	15	Développement
Semaine 13	22.05.2017	28.05.2017	15	Développement
Semaine 14	29.05.2017	04.06.2017	15	Développement
Semaine 15	05.06.2017	11.06.2017	15	Développement
Semaine 16	12.06.2017	18.06.2017	15	Développement
Semaine 17	19.06.2017	25.06.2017	24	*Crowdsourcing*
Examens	26.06.2017	02.07.2017	0	
Semaine 18	03.07.2017	09.07.2017	24	*Crowdsourcing*
Semaine 19	10.07.2017	16.07.2017	24	Évaluation
Semaine 20	17.07.2017	23.07.2017	24	Évaluation
Semaine 21	24.07.2017	30.07.2017	24	Rédaction
Semaine 22	31.07.2017	06.08.2017	24	Rédaction
Semaine 23	07.08.2017	13.08.2017	12	Remise
			360	

Annexe II : structure du résultat d'une requête du service Directions de Google Maps JavaScript API

Dans cette arborescence, les trois points (…) sont là pour souligner que le tableau contient potentiellement plusieurs éléments. Par exemple, le tableau routes[] contient plusieurs objets littéraux de type DirectionsRoute.



Annexe III : questions pour les entretiens des *focus groups*

Introduction : informations préliminaires aux participants

Saluer et remercier les participants.

But de l'étude : voir si l'application est utilisable. L'application est destinée à être utilisée par des personnes à mobilité réduite, mais aussi par leurs proches, c'est pourquoi vous êtes là. Le but de l'application est de permettre aux personnes à mobilité réduite (personnes en situation de handicap, personnes âgées, familles avec poussettes) une plus grande autonomie en ce qui concerne leurs déplacements.

Il n'y a pas de bonne ou de mauvaise réponse aux questions. Il s'agit de donner son opinion de manière libre et ouverte. Il ne s'agit pas de nous faire plaisir, mais de répondre franchement, afin d'améliorer l'application. Chacun doit parler à son tour, sans interrompre.

Nous devons **enregistrer** la session. Mais la retranscription sera anonyme.

1. Utilisabilité : application en général (navigation d'un écran à l'autre)

L'application vous semble-t-elle facile à utiliser (navigation, choix des langues) ?

Quelles difficultés avez-vous rencontrées dans l'utilisation de l'application ?

Qu'est-ce qui, selon vous, facilite l'utilisation de l'application ?

Quelles améliorations proposeriez-vous ?

2. Utilisabilité : saisie de points

L'application vous semble-t-elle facile à utiliser ?

Quelles difficultés avez-vous rencontrées dans l'utilisation de l'application ?

Qu'est-ce qui, selon vous, facilite l'utilisation de l'application ?

Quelles améliorations proposeriez-vous ?

3. Utilisabilité : calcul d'itinéraires

L'application vous semble-t-elle facile à utiliser ?

Quelles difficultés avez-vous rencontrées dans l'utilisation de l'application ?

Qu'est-ce qui, selon vous, facilite l'utilisation de l'application ?

Quelles améliorations proposeriez-vous ?

4. Intérêt de l'application

Pensez-vous que cette application puisse être utile ?

A qui ?

Pourquoi ?

Pensez-vous qu'elle apporte quelque chose par rapport à ce qui existe déjà ?

5. Crowdsourcing

Qu'est-ce qui pourrait vous inciter à saisir des informations dans le système ?

Conclusion de l'entretien

Arrêter l'enregistrement.

Remercier les participants.

Annexe IV : transcription des entretiens des focus groups

Introduction

La retranscription qui suit est fidèle. Cependant de petits bouts de phrases qui n'apportent pas grand-chose au débat ont été supprimés. Les interventions sont parfois paraphrasées, car un dialogue spontané ne donne pas toujours des phrases transcriptibles et parce que nous n'avons pas le temps nécessaire à une retranscription mot-à-mot. Parfois, pour éviter de multiplier les dialogues ou quand plusieurs personnes sont arrivées à une remarque ensemble, leurs contributions sont mises l'une après l'autre.

Les « […] » marquent un endroit où soit une démonstration a lieu, soit la conversation s'éloigne du sujet, soit plusieurs personnes parlent en même temps, soit plusieurs personnes développent ce qui a été dit sans ajouter d'éléments vraiment nouveaux de l'avis de l'auteur.

Les notes de bas de page sont des commentaires de l'auteur qui apporte son témoignage en tant qu'animateur de la séance. En effet, la retranscription des paroles exprimées ne rend pas toujours la pensée du locuteur puisqu'il manque les gestes et le contexte.

1. Groupe 1

1.1. Saisie des points

P3 : De manière générale, facile d'utilisation. Il y a des points de détails. Par exemple, quand on a choisi sa langue et qu'on peut sélectionner une adresse, la carte est vraiment trop petite.

Animateur : Il s'agit bien de cette carte ?[1]

P3 : Oui.

P2 : Surtout sur le téléphone, la carte est très petite.

P3 : Quand on a choisi un lieu, ça ne zoome pas sur le lieu. Si on veut être sûr que l'endroit sélectionné est le bon, il faut zoomer sur cette petite carte.

P2 : C'est fait plutôt pour un mobile ? Ou pour l'ordinateur ?

Animateur : Sur les deux. Les gens le feront souvent sur le mobile mais doivent pouvoir le faire aussi sur ordinateur.

P3 : Sur l'ordinateur, il y a le même problème, la carte est petite.

P4 : C'est frustrant cette petite carte.

P1 : Est-ce que sur l'ordinateur, on peut agrandir cette carte ?

P3 : Non on ne peut pas. On peut seulement zoomer et dézoomer.

P4 : On peut cliquer en bas sur le bouton carte.

P3 : Mais c'est pas la même.

P2 : Après avec le service itinéraire, la carte est assez grande.

P3 : Oui, mais c'est pas pareil.

P2 : Oui, mais il pourrait y avoir la même carte. Il n'y a que deux champs[2] dans la page, il y aurait donc la place pour agrandir. Le bas de l'écran est vide quand on entre une adresse.

P3 : J'imagine que les adresses dépendent de Google. J'ai eu ce problème. Je voulais ajouter un endroit, c'était une place qui n'existe pas dans Google. Du coup, j'ai mis le nom du village, j'ai créé un *point of interest*, j'ai entré le nom de la place, mais du coup l'emplacement n'est pas au bon endroit. Le point est situé au milieu du village à S***. Dans ce cas, il se trouve que ce n'est pas loin. Mais c'est pas très exact.

[1] Carte figurant en haut de la page position.php.
[2] Il y a, en plus de la carte, le champ de saisie de l'adresse et les boutons « Ajouter un nouvel endroit » et « Ajouter un endroit existant », ce dernier ouvrant une liste déroulante.

P4 : J'ai eu le même problème à L***, il a mis la patinoire que je n'ai pas trouvée à côté de la gendarmerie. Et après on ne peut pas changer l'adresse.

P3 : Je n'ai pas réussi à me localiser en cliquant sur « Ma position ».[3]

P2 : J'ai eu le même problème sur mon téléphone[4].

P4 : Sur l'ordinateur avec Firefox, il faut autoriser la localisation.

P2 : Je n'avais pas activé la localisation sur mon téléphone et le programme est resté bloqué à tourner. J'ai tout refermé. J'ai activé la localisation. Et là le programme m'a demandé si j'autorisais la localisation. Il faut donc l'activer, puis l'autoriser quand elle est utilisée dans l'application.

P3 : C'est sans doute ce que je n'ai pas fait, activer la localisation avant. Je n'ai eu aucun message d'information. Ça moulinait. Il devrait y avoir un message d'erreur qui dit d'activer la localisation.

[…]

P1 : Je croyais que ça allait dans les points existants dans la base de données. Quand j'ai ajouté le Technopôle qui y était déjà, il a mis un deuxième point.

P4 : Pour les points d'intérêt, il met seulement ceux qui se situent dans un rayon d'environ 200 m [5]. Je pensais trouver tous les points pour la commune de L***, si je tapais « L*** ». Du coup, je n'ai pas trouvé la patinoire, je l'ai donc rajoutée. Mais elle existe[6], je l'ai découvert après. Cela pourrait être mentionné que tous les points ne s'affichent pas.

[…]

P4 : Avez-vous noté qu'il faut pouvoir changer l'adresse d'un point ? Parce que la patinoire qui s'est mise au mauvais endroit, j'aimerais pouvoir la déplacer.

P2 : Quand on saisit la pente – car on peut jouer avec l'inclinaison du téléphone – ça serait bien qu'on puisse bloquer le chiffre, car comme il varie tout le temps on ne peut pas le saisir.

P3 : Il s'agit juste de voir si c'est supérieur à 10° ou pas.

[3] P3 possède un iPhone (iOS).
[4] P2 possède un Samsung (Android).
[5] Distance non vérifiée par l'auteur.
[6] P4 entend : elle existe déjà dans les POI de Google.

Animateur : Une personne ne sait pas naturellement si une pente est supérieure à 10° ou pas. C'est donc une aide pour savoir si c'est le cas, en mettant son téléphone parallèle à la pente. Mais au moment de la saisie, le téléphone n'est pas forcément parallèle à la pente.

P4 : Je n'ai pas vu cet élément.

[…][7]

P2 : Ah d'accord. C'est indicatif. Moi je pensais qu'il sauvegardait la pente exacte directement.

P4 : Mais 10°, c'est rien du tout !

Animateur : Oui mais en fauteuil roulant…

P3 : Donc, il y a trois critères qui sont des difficultés : les obstacles, le trottoir trop étroit et la pente et trois facilités[8]. J'ai voulu ajouter une route piétonne. Sauf qu'elle ne remplit aucun des six critères. J'ai d'abord voulu cocher « trottoir < 1.5 m », car il n'y avait pas de trottoir. Mais ce n'est pas une difficulté, puisque c'est une route piétonne. Je ne sais pas si c'est bien de la saisir, mais c'est impossible. Aurais-je dû mettre « entrée sans marche » ? Est-ce que la liste de critères va évoluer ?

[…]

P2 : Ne serait-il pas possible d'entrer juste un commentaire, avec une appréciation positive ou négative, afin d'avoir la couleur du point, rouge ou vert, sur la carte ?

P4 : Je trouve les trois boutons en bas peu visibles.

Animateur : Ces boutons-ci[9] ?

P4 : Oui.

P2 : Moi au contraire je les aime bien.

P1 : Si on agrandit la carte, on les verra encore moins.

[7] L'animateur a voulu montrer cet élément (calcul de la pente sur le bouton « Pente > 10° » à P4. Mais il n'apparaît pas sur son téléphone Samsung (Android Lollipop) sous Firefox. Après contrôle, il apparaît bien sous Google Chrome.

[8] Parking pour handicapés, toilettes pour handicapés et accès sans marches.

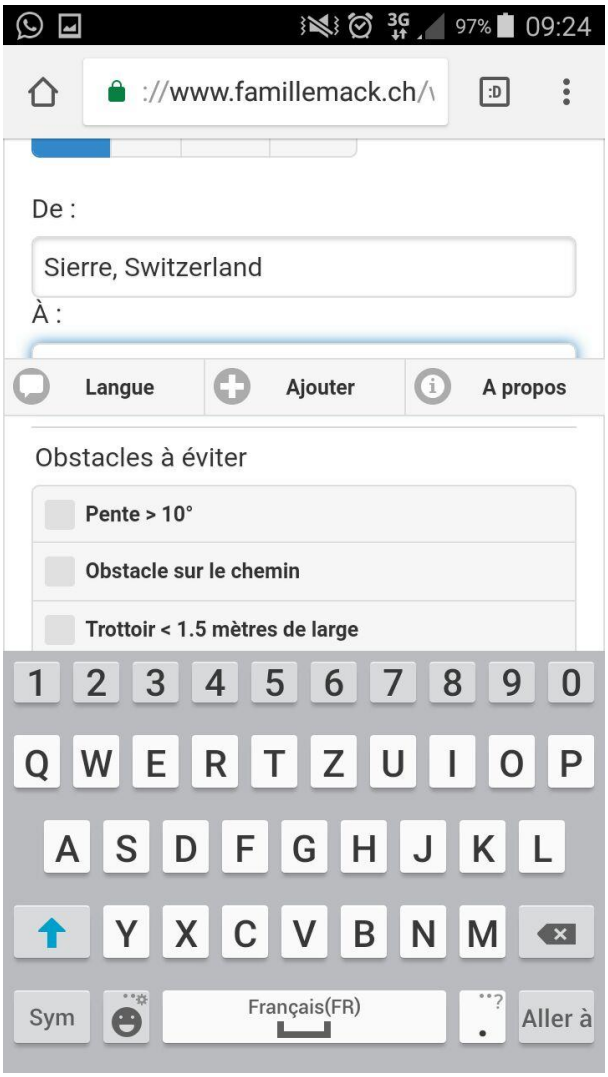
[9] « Langue », « Carte » et « A propos » quand on est à la saisie des points dans position.php.

P2 : Oui c'est vrai si on agrandit la carte.

[…]

1.2. Carte et saisie des itinéraires

P2 : Quand je saisis les itinéraires, la 2e fois, j'ai justement ces trois boutons qui remontent dans la page. Ils arrivent sur le formulaire et la saisie devient impossible. Je peux envoyer une capture d'écran :



P3 : Quand on saisit un itinéraire et qu'on scrolle, la carte en bas se déplace au lieu que ce soit le tout. Parfois sur certains sites, c'est écrit qu'il faut utiliser deux doigts pour bouger la carte. Ainsi on peut scroller avec un doigt, on peut, et si on veut bouger la carte on utilise deux doigts.

P1 : Sur l'ordinateur, on ne peut pas prendre de photo avec la webcam. On peut charger une photo qu'on a déjà, mais pas prendre une photo.

P3 : Par contre ce qui est bien est que quand on fait une photo avec le téléphone, elle ne s'enregistre pas dans le téléphone. Il existe des applications qui le font et c'est embêtant, ce sont souvent des photos qu'on n'a pas envie de garder.

P2 : Tous les critères sont affichés [quand on clique sur un marqueur] et en noir ce sont ceux qui sont activés. Est-ce que ça ne serait pas plus lisible d'afficher que ceux qui sont activés, sans voir les autres ?

P4 : Oui, pourquoi ne pas activer que ce qui ne va pas ?

P1 : Ah, ce qui est grisé, il n'y a pas ?

P3 : C'est pas très clair.

P1 : Je n'avais pas compris, ça.

Animateur : Est-ce que ça devrait être grisé plus clair ?

P2 : Non, j'enlèverais. On gagnerait de la place, en plus.

P3 : C'est vrai qu'à un moment je me suis dit : mince, j'ai saisi faux quand je suis retourné voir si mes points étaient bien corrects.

P2 : Ou alors plus clair, si on veut absolument garder les critères.

P3 : Ou alors plus clair, oui, mais vraiment beaucoup plus clair.

P2 : Le contraste n'est pas suffisant.

P1 : J'ôterais totalement.

P3 : Je ferai aussi une petite aide, que quand on appuie dessus ça affiche ce que c'est. Ou quand on passe la souris dessus à l'ordinateur.

P3 : Quand on va dans le détail d'un lieu, les boutons ne sont à mon avis pas dans le bon ordre : je mettrais le bouton « Supprimer » en dessous de « Modifier », ou à la limite dans « Modifier » : quand on modifie, on a la possibilité de supprimer. J'ai l'impression que le bouton « Supprimer » est un peu trop accessible. En scrollant pour trouver le bouton « modifier », on risque de cliquer par erreur sur « Supprimer ». C'est très vite arrivé.

P2 : Tu as raison.

P4 : Y a-t-il une demande de confirmation, quand on supprime ?

Animateur : Oui.

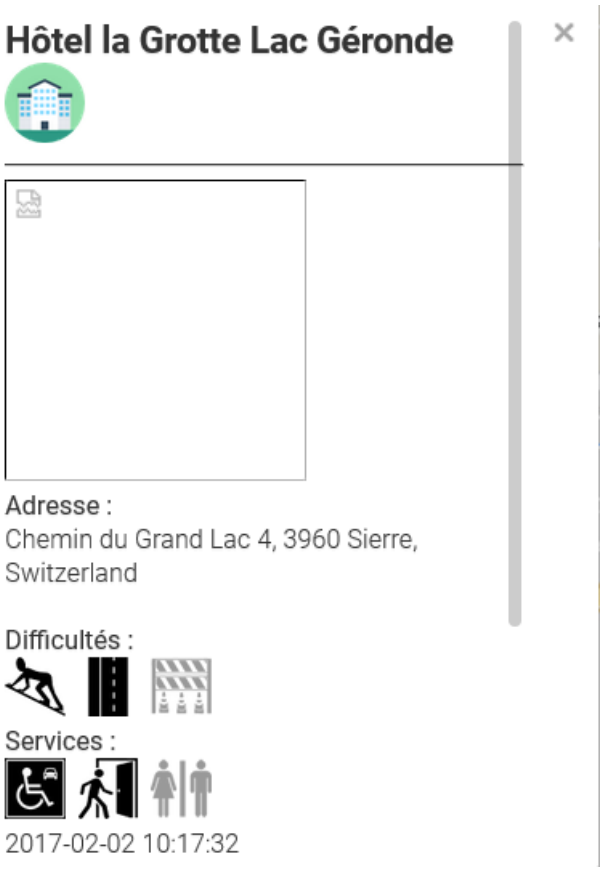
P2 : Quand on ajoute un point, qu'on clique sur « Ajouter un endroit existant » et qu'on scrolle dans la liste des endroits existants proposée, le titre « Ajouter un endroit existant » disparaît, il est scrollé avec la liste.

P3 : Il y a des images qui ne s'affichent pas.

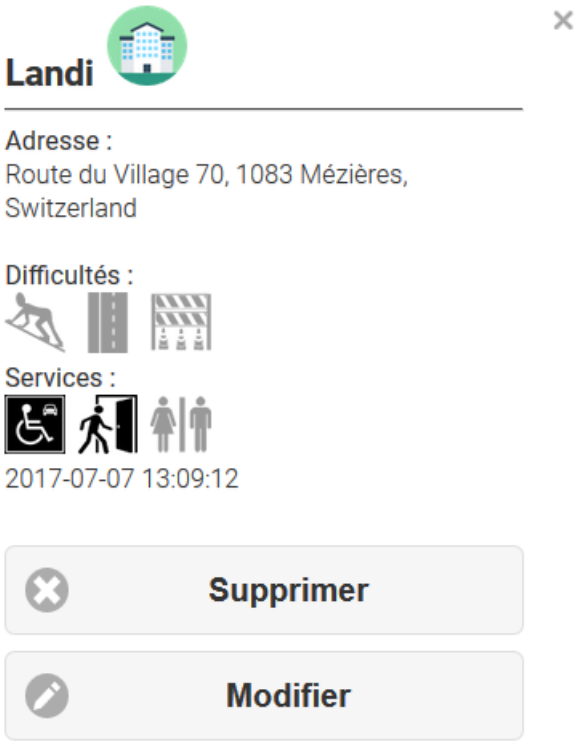
Animateur : Toutes les images antérieures ne sont pas accessibles dans cette version de démonstration.

[…]

P3 : Donc quand il n'y a pas d'image, ça affiche le cadre avec la petite image cassée ?



Animateur : Oui quand une image existe mais que le système ne la trouve pas… Quand aucune image n'a été enregistrée, ça n'affiche rien :



P3 : Alors quand il ne trouve pas l'image, ce serait mieux d'afficher rien plutôt qu'un cadre vide.

P2 : Quand on veut insérer la photo, il y a un bouton et l'image qui s'affiche. Il n'y a pas moyen de combiner les deux ? Le bouton est aussi grand que l'image.

P3 : Moi ça ne m'a pas gêné.

P2 : Ça ne m'a pas gêné. Mais n'y aurait-il pas moyen que la photo occupe tout cet espace ?

Plusieurs : Le bouton pourrait être plus petit.

P3 : Il faudrait peut-être faire une analyse ergonomique. [10]

P1 : Pour les mal-voyants, c'est mieux d'avoir un grand bouton.

P3 : D'ailleurs, est-ce que ça a été pris en compte, les mal-voyants ?

P1 : Je pense que oui, car tous les boutons, quand on ajoute un nouvel endroit, sont énormes.

P3 : C'est vrai que souvent les applications avec des gros boutons, c'est plus évident à utiliser.

[…]

P3 : Vous avez pensé aux daltoniens ? Je ne sais pas si vert et rose est visible pour eux.

P2 : Il faudrait mettre des carrés et des ronds.

[…]

P2 : Pour les icônes qui apparaissent en gris, peut-être faudrait-il faire un gris plus clair.

1.3. Intérêt de l'application

P2 : Je pense que ça peut être très utile.

P3 : Ça peut être bénéfique. Par exemple, quand j'ai photographié cette place, à S***, où on peut jouer à la pétanque, pique-niquer : il y a plein de gravier devant. Du coup, ça peut poser un problème d'accessibilité, alors que c'est un endroit sympa. Si on a un moyen comme ça de contrôler et si ça pose problème, on peut décider d'aller ailleurs. Ou alors on prévoit du matériel en conséquence.

P2 : Du coup, je trouve qu'il manque des critères, par exemple s'il y a des escaliers.

Animateur : Un escalier se marque en cochant « obstacle », puis on peut mettre un commentaire et prendre une photo.

P2 : Quand je pense à mes grand-parents qui ont de la peine à marcher, quand on va dans un endroit qu'on ne connaît pas, on peut choisir la route que l'on va prendre et savoir s'ils peuvent y aller tout seul ou pas.

[10] Ton ironique.

P3 : Je trouve ça plus utile pour les points d'intérêt que pour les routes. Après je ne suis pas dans la situation où j'ai besoin de contrôler un itinéraire.

P1 : Moi si je suis en chaise roulante, je souhaiterais pouvoir dire je vais à la gare et que ça m'affiche les points.

P3, P2 : Mais c'est ce que ça fait !

[…][11]

P1 : Alors là ça va pas. Je n'ai pas trouvé le bouton quand j'ai testé. J'ai cliqué en bas. J'ai testé sur l'ordinateur. On ne va pas cliquer là-haut [sur « Service itinéraire »] !

P3 : Moi je ne l'ai pas vu tout de suite. Le logo n'est peut-être pas le bon pour « Service itinéraire ». Dans Google, le logo pour les itinéraires est celui-là :  . Celui-ci [], c'est soit la boussole, soit les localisations.

P1 : Ou alors le texte pourrait être : « Entrez un itinéraire ».

[…]

P4 : Est-ce que l'itinéraire évite les obstacles ?

Animateur : Non, il indique où ils sont.

P4 : Mais est-ce qu'il y alors une option : prendre une autre route ?

P3 : Il serait bien de pouvoir dire qu'un critère est critique : je ne peux pas gérer ce type d'obstacle.

P2 : J'y pense maintenant : est-ce qu'il y aurait moyen de mettre un numéro de téléphone dans les POI ? Pour contacter quelqu'un qui vient aider si l'accès est difficile.

P3 : J'imagine que ça dépend des données dans Google. On pourrait mettre le numéro dans les commentaires.

P2 : Par exemple dans les gares CFF, on doit avertir quand on prend le train en fauteuil roulant, pour que quelqu'un soit là pour aider.

[11] Démonstration…

Animateur : Parfois dans un restaurant on peut entrer par la porte de derrière, mais elle est verrouillée, il faut avertir pour qu'on nous ouvre.

P3 : Les gens peuvent mettre cette information, mais il faut les y faire penser.

1.4. Crowdsourcing

P3 : C'est compliqué. Ce sont des situations auxquelles on ne va pas s'intéresser, tant qu'on n'y est pas sensibilisé parce qu'on connaît quelqu'un.

P3 : Faire des prix.

P2 : Mettre des Pokemons go…

(rires)

P2 : Le problème est que les gens vont peut-être saisir n'importe quoi. Je ne suis pas convaincue.

P3 : Si je joue à un jeu, je joue à un jeu, je ne saisis pas des données.

P2 : À la limite ça peut encourager ceux qui sont motivés.

P3 : Des bons de dix pour cent chez Sun Store.

P1 : La bonne idée, c'était d'impliquer les facteurs. Là ils ont joué le jeu. On pourrait pas voir avec DPD etc. ?

P3 : Ils sont trop stressés. Déjà que les facteurs… Alors ils risquent de saisir faux.

P1 : On pourrait impliquer les employés des offices de tourisme. Quand ils ont des périodes mortes, ils recensent les activités. Ça serait un plus pour la station d'avoir les informations. Et aussi les organisateurs de manifestations.

P3 : Ou les CMS, qui visitent les gens et qui ont cette fibre sociale.

P3 : On pourrait aussi faire une campagne de sensibilisation auprès du grand public pour susciter des vocations.

P2 : J'avais lu le tutoriel avant d'essayer.

P3 : On aurait peut-être dû essayer sans le tutoriel. Normalement, ce doit être suffisamment intuitif.

2. Groupe 2

La conversation ayant naturellement commencé avant le début de l'enregistrement, ce dernier débute de manière décousue…

2.1. Saisie de points, carte et calcul d'itinéraire

P6 : [Pour la saisie des routes,] je trouve qu'on devrait pouvoir aller sur la carte, clic, sélectionner un segment de route avec le doigt.

P5 : [Dans les InfoWindows,] il faudrait que les critères absents soient barrés, ou alors marquer de manière plus évidente ce qu'il n'y a pas, plutôt que juste grisés. Il n'est pas évident qu'ils sont absents.

P7 : C'est quoi la différence entre les points roses, verts et tout ça ?

Animateurs : Les rouges sont ceux qui ont un des trois critères négatifs. Les autres sont verts.

P5 : Il devrait y avoir quelque part un guide d'utilisation au début quand on arrive sur l'application : pourquoi les points sont rouges, pourquoi ils sont verts.

P6 : Je suis complètement fan de ce truc de la pente[12].

P7 : On devrait pouvoir mettre des commentaires.

Animateur : Mais on peut, sur les points.

P5 : On devrait pouvoir mettre : déconseillé. Par exemple, au restaurant des A***, si on veut rentrer avec sa chaise roulante, on peut. Mais après c'est vraiment pas adapté.

P6 : Par exemple les bateaux de la CGN, on peut aller avec une chaise roulante, j'ai essayé avec celle de ma grand-mère, mais il faut porter la grand-mère de 100 kg sur le bateau…

P6 : Au début j'ai confondu « ajouter un nouvel endroit » avec « ajouter un endroit existant ». Au lieu de « ajouter un endroit existant », il pourrait être inscrit quelque chose comme « ajouter un point qui existe sur Google » ou « chez Google », je ne sais pas. Pour moi ce n'était pas clair au début.

[…]

[12] Fait que l'inclinaison du téléphone en degrés s'affiche sur le bouton « Pente > 10° ».

P6 : Le fait de pouvoir prendre une photo est génial. Ça permet de voir comment est vraiment la situation.

P5 : J'ai de la peine à définir quand on met « POI » ou « route ». Quand on met « route », c'est quand même un point.

Animateur : Oui c'est un point le long d'une route.

P5 : Donc si je veux signaler les escaliers à côté de la banque cantonale, je mets que c'est une route ? Mais alors comment est-ce que j'inscris les escaliers ?

Animateur : Il faut mettre « obstacle », inscrire sous commentaire qu'il y a un escalier. Et prendre une photo de l'escalier.

[A ce stade de l'entretien, l'auto-complétion ne fonctionne plus dans les champs]

[…]

P6 : Est-ce que l'application permet aussi de localiser les numéros de plaque des c******* qui se parquent sur les places pour handicapés ? Qu'ils se fassent un peu lyncher.

(rires)

P5 : Est-ce que ça marche aussi pour les randonnées ?

Animateur : On peut aussi analyser ces chemins. Mais le but de l'application, c'est d'aider les personnes à mobilité réduite, les familles avec poussettes. Pas de développer le tourisme rural.

P7 : Parce qu'une poussette, c'est un handicap ?

Animateur : Dans les escaliers, oui.

P6 : Parfois, on n'est pas directement à l'endroit qu'on veut saisir. Il serait bien de pouvoir choisir un autre point pas très loin, en cliquant. C'est du pinaillage.

Animateur : Non c'est intéressant. On est là pour ça.

P6 : Serait-il possible de mettre la carte plus grande ?

P7 : La touche « Supprimer », c'est énorme. On peut supprimer sans autre les points ?

Animateur : Oui, mais il y a une demande de confirmation.

P7 : Mais ça pourrait être le grand m****ier si quelqu'un supprime des tas de points.

Animateur : D'une part les points supprimés sont conservés quelque part dans le système et d'autre part quand le système sera terminé et que les gens pourront s'inscrire, on ne pourra supprimer que les points qu'on a soi-même saisis.

P5 : Concernant la première carte [celle de l'écran position.php]. C'est quoi ce vide [en bas de l'écran position.php]. On pourrait avoir une carte plus grande et moins de vide.

P6 : Il faudrait mettre « Ajouter un nouvel endroit » et « Ajouter un endroit existant » juste sous la recherche, et puis la carte dessous.

P5 : C'est pourquoi « A propos » ?

Animateur : Cliquez !

P5 : Eh il n'y a même pas ton nom !

[…]

P5 : On pourrait mettre là : les points rouges sont ceux à éviter, etc.

P7 : Un petit texte succinct pour expliquer les grandes lignes, ce ne serait pas mal.

[…]

P6 : Concernant la carte trop petite, si le menu déroulant qui s'ouvre avec les endroits existants couvre la carte, ce n'est pas grave. On n'en a plus besoin.

[…]

P6 : Je pense que même si Google a un monopole désagréable, c'est intéressant de prendre eux, parce qu'ils ont des tas de données notées.

P7 : Donc si on va en vacances au Cambodge, on peut mettre le temple machin est accessible, ou le temple machin n'est pas accessible ?

P5 : Ce qui est vraiment top, c'est qu'on peut mettre une photo.

P7 : Oui car celui qui est concerné peut estimer la problématique.

P6 : Mais en fait, le Zoo de S***, je l'ai mis ici !

P5 : Donc on est obligé de le supprimer pour le remettre juste ?

[…]

2.2. Crowdsourcing

P5 : [Pour motiver les gens, il faut] leur donner de la thune.

P6 : Je trouverais bien qu'on voie que c'est utile pour les gens. Je ne sais pas comment. Qu'on soit informé si quelqu'un a observé un des points saisis.

P5 : Ou un petit forum.

P6 : À part l'altruisme, l'empathie…

P7 : Les gens dont ça valorise le local ont intérêt à saisir. Ou alors quand on a saisi 500 ou 100 points, on reçoit une récompense.

P5 : Ne serait-ce qu'un classement. Le classement ça marche bien.

P5 : Il serait bien qu'il y ait un forum ou la possibilité de mettre un petit message pour que les gens puissent se causer.

P7 : Une interaction c'est intéressant. La photo, c'est bien. Mais si les gens veulent échanger entre eux, c'est bien aussi.

P5 : Ou alors un forum : « Êtes-vous déjà allés au Mont-Blanc, y a-t-il une difficulté ? ». « Oui, il y a plus de 10° de pente et pas de toilettes pour handicapés. »

P7 : Je pense que les personnes concernées seront motivées à contribuer.

[…]

P5 : Je continue à penser que pouvoir poser une question : je dois aller au café XY, quelqu'un y est-il déjà allé ?

[…]

P6 : Je reviens à ma grand-mère en chaise roulante, chaque fois qu'on allait quelque part il fallait se renseigner. Cette application aurait été très utile. Et c'est important d'y penser. Je viens de faire un site Internet pour notre établissement, je n'ai pas du tout pensé à préciser qu'il y a un accès pour les handicapés.

P7 : Ça peut prendre des années jusqu'à ce que tout soit recensé partout.

P6 : Quand Google a commencé à prendre des photos de rue, ils ont dû se dire la même chose. Mais ils y sont arrivés.

Animateur : Oui mais ils disposent de plus de milliards que nous.

P7 : Si tout est rempli, il y aura une forêt de petits points.

P5 : Oui mais après tu peux zoomer.

P6 : Dans Booking, c'est pareil. Mais quand on est loin, il y a un petit truc avec par exemple 30 points.

P5 : Les randonneurs pourraient être intéressés. Ils aiment aussi savoir si ça monte, s'il y a du gravier, de la boue.

P7 : Pour les familles aussi ça peut être utile. Même sans parler de pousse-pousse, rien qu'avec des enfants.

P6 : Je pense que ce serait une autre application pour les enfants. Il ne faut pas s'écarter. Ici c'est pour les personnes en situation de handicap.

Annexe V : synthèse des entretiens des *focus groups*

Application

Fonction	Remarques	Participants	Commentaire
Naviguer dans l'application	Trois boutons en bas peu visibles	P4 Les autres ne sont pas d'accord	-
Ajouter des informations	Quand géolocalisation pas active pour le navigateur et qu'on clique sur « Ma position », le programme tourne sans avertir qu'il faut activer.	P1, P2, P3, P4 (P5, P6 et P7 ont été avertis de le faire)	Nous avons ajouté une fonction qui affiche un message d'erreur s'il y a un problème dans la géo-localisation. Cette fonction affiche un 'toast'
	Carte trop petite Le vide en bas ne sert à rien (il faudrait remonter les boutons « Ajouter un nouvel endroit » et « Ajouter un endroit existant » ; si la carte disparaît quand on clique sur ajouter un endroit existant, c'est pas un problème)	P1, P2, P3, P4, P6, P5 P5, P6 P6	La carte avait été réduite suite à la demande des facteurs qui ont récoltés les premiers POI. En effet ils disposaient de petits téléphones et la liste de points existants ne présentait que 2 POI à la fois
	« Ajouter un endroit existant » : quand on scrolle dans la liste, ce titre disparaît	P2	Comme pour la taille de la carte, il s'agit de gagner une ligne dans la liste quand on a un petit téléphone.
	Différence entre « Ajouter un nouvel endroit » et « Ajouter un endroit existant » pas comprise tout de suite	P1 avait compris un point existant dans WE-MAP, pas dans Google	Dans une version précédente, c'était écrit « Ajouter un POI » et c'était encore moins clair.

Fonction	Remarques	Participants	Commentaire
Ajouter des informations (suite)	« Ajouter un endroit existant » : l'endroit doit se trouver à 200 m. Si on a saisi le nom d'une localité, certains éléments sont à plus de 200 m du centre	P3, P4	Cette limite a été ajoutée pour éviter que la liste de POI ne comporte des dizaines de lignes ce qui rend la recherche plus difficile. Le plus souvent on sera sur place et le POI recherché sera à moins de 200 m.
	Souhaitent pouvoir déplacer un point après coup	P3, P4	-
	Ajout d'un endroit qui ne remplit aucun critère	P3	Cela a été ajouté entre temps par un autre membre de l'équipe
	Ça serait pratique de pouvoir cliquer sur la carte pour ajouter un point	P6	-
	On devrait pouvoir mettre « déconseillé ». Il y a beaucoup d'endroit où on peut aller (restaurant Amici, bateau de la CGN) mais où c'est juste pas pratique	P5, P6, P7	Ce genre d'information peut être saisi sous commentaire
	Les photos c'est génial, ça permet de juger vraiment la situation	P6, approuvé par P5	-
	Très bien que les photos ne s'enregistrent pas sur le mobile	P3	-
Saisie itinéraires	Pas trouvé le bouton « Service itinéraire ». Le texte pourrait être « Entrez un itinéraire » Logo pas le bon, devrait être  au lieu de	P1 P3	Nous avons mis une icône personnalisée.
	Pouvoir calculer un itinéraire qui contourne l'obstacle	P4, P3	Ce sera peut-être l'objet d'un développement ultérieur.

Fonction	Remarques	Participants	Commentaire
Saisie itinéraires (suite)	Quand on saisit une 2e fois, les 3 boutons du bas remontent dans le formulaire	P2	Ce *bug* est difficile à reproduire. Il disparaît suivant ce qu'on fait. Nous gardons son éventuelle correction pour plus tard
Fenêtres d'information	Les éléments absents en gris : ce n'est pas clair qu'ils sont absents (devraient être absents) (éventuellement plus transp.) (logo barré)	P1, P2, P3, P4, P5, P6 P1, P2, P3 P2, P3 P5	Nous avons mis une opacité de 20 % au lieu de 40%
	On devrait pouvoir passer la souris (hover) ou cliquer avec le doigt pour avoir une explication des critères	P3	Nous avons ajouté des tooltips.
	Le bouton « Supprimer » devrait être en-dessous de « Modifier », voire atteignable que dans « Modifier » pour P3. On risque de cliquer dessus en scrollant. (mais il y a une confirmation)	P2, P3	Nous avons changé l'ordre
	Photo enregistrée introuvable : affiche une fenêtre vide (alors que pour une photo pas prise, il n'y a rien)	P3	Ce phénomène ne s'est produit qu'avec la version de démonstration, car les photos prises par les postiers n'avaient pas été mises en ligne. Il ne se produira jamais.
Carte	Qu'en est-il des daltoniens pour les points verts et roses (les participants voient des points roses, pas rouge)	P3, P2	-
	Pas compris la différence entre points verts et roses ; il faudrait un mode d'emploi, la première fois qu'on arrive sur l'application	P7, P5	Nous avons créé un tutoriel de cinq écrans, avec texte et images en fonction de la langue.

Intérêt de l'application

Les participants du groupe G1 estiment que l'application peut être très utile.

P2 : l'application serait utile pour ses grand-parents qui ont de la peine à se déplacer. Cela permettrait de savoir si un endroit est adapté ou non, s'ils peuvent s'y rendre seuls, la route à prendre pour y aller.

P3 : l'application, avec les photos, permet de se rendre compte si un endroit est adapté.

Les participants du groupe G2 soulignent l'intérêt des photos pour estimer la difficulté.

P6 : sa grand-mère aujourd'hui décédée était en fauteuil roulant. Il fallait toujours se renseigner pour savoir si un restaurant ou un autre endroit était accessible. L'application aurait simplifié la vie.

Crowdsourcing

P3 : faire des prix

P2 : Gamification va encourager que ceux qui sont déjà motivés ; risque que les gens saisissent n'importe quoi pour gagner

P1 : Impliquer les employés des offices de tourisme quand ils ont des temps creux ; les organisateurs de manifestation…

P5 : payer les gens

P7 : récompense pour *X* saisies ; les personnes concernées (handicapés ou personnes dont ça valorise le local) seront motivées à contribuer

Annexe VI : modifications apportées à l'application

Le tableau ci-dessous résume de manière succincte les modifications qui ont été apportées à l'application tout au long de ce travail de Bachelor. La dernière colonne renvoie au(x) chapitre(s) de ce mémoire dans le(s)quel(s) la modification a été apportée. En effet, certaines modifications faisaient partie des objectifs de ce travail, en lien avec le calcul d'itinéraires. D'autres modifications ont été apportées dans la partie développement suite aux discussions avec les responsables du projet mais n'étaient pas directement liées au calcul d'itinéraires. Enfin, suite aux entretiens de groupes, il a été décidé d'apporter des améliorations.

Réalisation	Détails	Fichiers impactés	§ du mémoire
Adaptation du menu de calcul d'itinéraire.	Un menu rudimentaire était déjà inséré dans l'application	map.php	2.3.1
Auto-complétion des champs « De » et « À »	Quand l'utilisateur saisit une adresse, les données disponibles de Google s'affichent.	route_service.js	2.3.2
Calcul d'itinéraires	Utilisation du service Directions de Google Maps API Directions	route_service.js	2.3.3.1 2.3.3.2 2.1
Affichage de l'itinéraire	Création d'une google.maps.polyline et affichage sur la carte	route_service.js	2.3.3.2 2.1
Traitement des marqueurs sur la carte		route_service.js map.js	2.3.3.3 2.2 2.3.3.5
Centrage de la carte	La carte doit afficher tous les trajets complètement	route_service.js	2.3.3.4
Sélection d'un des chemins par l'utilisateur	L'utilisateur doit pouvoir cliquer sur un des chemins. Il apparaît en bleu, les autres en gris et une étiquette afficher la longueur de ce trajet.	route_service.js	2.3.3.6
Réinitialisation	Cette fonction supprime tous les itinéraires de la carte et rétablit tous les marqueurs, dans leur couleur d'origine	route_service.js	2.3.4
Ajout de marqueurs en début et fin de trajet	Cela rend l'application plus « Google-compatible »	route_service.js	2.4
Infowindows : déplacement de l'image	Mise en tête d'une colonne unique plutôt que seule dans une colonne à droite	map.js	2.5

Réalisation	Détails	Fichiers impactés	§ du mémoire
Infowindows : affichage des critères absents	Nous avons affiché en grisé les critères absents et nous avons différencié les catégories « Difficultés » et « Services ». Après l'analyse, nous avons encore éclairci les critères absents	map.js	2.5 4.5.3
Messages liés aux erreurs de la géolocalisation	Il n'y avait pas toujours un message	position.js	4.5.1
Modification de l'icône de calcul d'itinéraire	Afin qu'elle ressemble à l'icône Directions de Google	map.php map.css	4.5.2
Infowindows : Ajout de tooltips sur les critères	Quand on clique sur les icônes des critères ou qu'on passe le pointeur dessus, un texte s'affiche	map.js	4.5.4
Infowindows : interversion des boutons « Modifer » et « Supprimer »	Selon la suggestion d'un testeur, il y a risque lorsqu'on scrolle avec le doigt de cliquer sur « Supprimer »	map.js	4.5.5
Tutoriel	Ajout d'un tutoriel bilingue (potentiellement multilingue) de cinq écrans	tutorial.php	4.5.6

Déclaration de l’auteur

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après : Nicole Glassey Balet, Zhan Liu.