PawAtlas

Application web progressive pour la gestion d'animaux de compagnie

TRAVAIL DE BACHELOR

PAUL RICCI

Septembre 2024

Supervisé par :

Prof. Dr. Jacques PASQUIER–ROCHA
Software Engineering Group

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Software Engineering Group
Department of Informatics
University of Fribourg
(Switzerland)

Remerciements

Par ces mots, je souhaite vivement remercier le Prof. Dr. Jacques Pasquier-Rocha pour son suivi, sa souplesse et ses précieux conseils lors de la réalisation de ce travail. Je souhaite aussi remercier de tout cœur ma compagne et ma famille pour leur soutien infaillible dans l'accomplissement de ces études. Mes remerciements s'adressent également à mon meilleur ami qui m'a mandaté pour ce projet à travers son entreprise E-ProShop Sàrl.

Résumé

Ce travail résume les étapes nécessaires à la création d'un prototype d'application web progressive avec le framework Angular et la plateforme de développement Firebase.

Il répond à la demande de l'entreprise E-ProShop Sàrl qui souhaite commercialiser une application de gestion d'animaux de compagnie. Celle-ci permettrait aux utilisateurs de se créer un profil personnalisé, puis de le faire pour leurs compagnons. Une carte interactive leur permettrait d'ajouter et de consulter des lieux d'intérêts, comme des vétérinaires ou des centres animaliers, mais aussi des dangers tels que des zones toxiques ou interdites aux animaux.

Dans ce travail de Bachelor, les cas d'utilisation liés à la carte interactive ont été implémentés dans un prototype. De plus, un système d'authentification a été mis en place tout comme une base de données de type NoSQL.

L'implémentation a été facilitée grâce à Angular et ses automatisations, mais ce travail écrit propose une analyse des apports du framework, notamment ceux concernant la transformation d'une application web en une *« Progressive web app PWA »*.

Afin de gérer efficacement les opérations asynchrones liées au développement web, le code présenté propose une approche qui repose sur des concepts de programmation réactive à travers l'utilisation d'une librairie dédiée.

Le prototype réalisé a été rapidement déployé pour qu'il puisse être testé. Pour des raisons budgétaires, la suite de développement Firebase a été utilisée car elle est gratuite actuellement. Ce travail détaille son utilisation et les conséquences liées à ce choix.

Pour finir, plusieurs améliorations et perspectives sont mentionnées à la suite de la conclusion.

Mots clés: Application Web Progressive, PWA, Angular, Firebase, Programmation réactive.

Table des Matières

Liste des Figures

Liste des Codes source

1 Introduction

1.1 Motivations et objectifs

Après une carrière d'une dizaine d'année dans le domaine du médico-social, plus particulièrement dans le maintien à domicile de personnes âgées, j'ai décidé de me réorienter professionnellement et d'entreprendre un Bachelor en Informatique de Gestion. Durant mes études, mon meilleur ami qui détient un Master en Gestion d'entreprise m'a contacté avec un projet d'application de gestion d'animaux de compagnie. À travers son entreprise E-ProShop Sàrl et avec l'aide de son assistant, il a réalisé un cahier de charges qu'il m'a transmis afin de savoir si je souhaitais débuter cette aventure.

C'est de cette manière que je me suis tourné vers le Prof. Dr. Jacques Pasquier-Rocha pour lui proposer ce projet et réaliser un prototype qui fait ainsi l'objet de ce Travail de Bachelor.

Ce mandat me motive dans le sens où il rejoint les différents objectifs que je me suis fixés dans ma reconversion professionnelle en vue de devenir informaticien de gestion. D'une part, il me permet de me mettre en situation en me confrontant aux désirs de mandataires en termes de développement de logiciels. D'autre part, il me permet d'approfondir mes compétences en programmation.

Lors de nos discussions avec l'entreprise E-ProShop Sàrl, l'objectif était de développer une application qui évitera les différents magasins de distribution Apple et Android. Ces derniers prélèvent une part importante sur les résultats financiers. C'est ainsi que nous nous sommes tournés vers le framework Angular et le développement d'une application web progressive, utilisable autant sur mobile que sur un ordinateur. Apprendre à utiliser ces technologies me motive vivement car c'est dans le domaine du web que je souhaitais me réorienter professionnellement.

Pour finir, les animaux de compagnie ne sont pas une source centrale de motivation pour ma part. Comme je le répète souvent, tout projet nécessite un prétexte pour parcourir un chemin bien plus important que son résultat.

1.2 Structure du rapport

Chapitre 1 : Introduction

Ce dossier débute par un chapitre détaillant mes motivations et les objectifs de ce travail. La présente structure du rapport y figure, tout comme les conventions liées à sa rédaction.

Chapitre 2 : Fonctionnalités et modélisation

Ce chapitre présente le contexte et l'origine du projet. Il mentionne le cahier de charges qui m'a été confié ainsi que les choix réalisés pour le développement du prototype. Les cas d'utilisation retenus sont détaillés et une modélisation de la base de données y figure.

Chapitre 3 : Point de vue utilisateur

Cette partie comprend les résultats obtenus d'un point de vue utilisateur. L'ensemble des fonctionnalités présentes dans le prototype réalisé sont documentées.

Chapitre 4 : Point de vue développeur

L'implémentation du prototype est abordée dans cette partie. Des extraits de code sont expliqués et une analyse des éléments centraux est effectuée.

Chapitre 5 : Conclusion

La conclusion de ce travail est présentée dans le dernier chapitre. De plus, des améliorations et perspectives pour l'avenir du projet sont discutées.

1.3 Conventions et notations

- La langue française a été choisie pour la rédaction de ce Travail de Bachelor, toutefois, l'ensemble du code ainsi que ses commentaires seront rédigés en anglais.
- Sans pour autant négliger son importance et afin d'alléger la lecture, ce travail n'est pas rédigé dans un langage épicène. Si parfois des termes sont mentionnés au masculin, ils ont été pensés et sont à interpréter de manière neutre et non genrée.
- Toute figure et tout code source est inscrit dans sa liste respective selon l'ordre d'apparition.
- Le code source apparait dans le texte de la manière suivante :

```
1     // This is a comment...
2     export function foo() {
3       console.log("foo");
4     }
```

Code 1 : Exemple de code source

- L'intégralité du projet se trouve dans un dossier privé sur GitHub. Étant donné que cette application sera mise en production et par la suite éventuellement monétisée, certains éléments du code restent confidentiels.

- Malgré un dossier GitHub privé, c'est avec plaisir que je répondrai à toutes questions concernant l'implémentation et partagerai le code à celui qui en fait la demande. Les fonctionnalités principales de l'application sont détaillées et des extraits sont disponibles dans ce travail écrit.
- De manière générale, la première personne du pluriel est utilisée pour la rédaction de ce travail. Cependant, il a été réalisé de manière individuelle et la première personne du singulier reste utilisée pour les parties qui concerne directement son auteur.

2

Fonctionnalités et modélisation

2.1 Contexte

Dans ce chapitre, nous aborderons plus en détail ce projet, son contexte et son origine. Puis, nous analyserons les cas d'utilisation du prototype à réaliser ainsi que la modélisation de la base de données.

2.1.1 Le projet

Origine du projet

Ce projet a vu le jour pour donner suite à la demande de mon meilleur ami dans le cadre de son entreprise E-ProShop Sàrl [1]. Diplômé d'un Master en Gestion d'Entreprise, il a créé cette société en s'orientant dans le développement de boutiques en ligne et dans le consulting. Nous pouvons relever d'ailleurs que son site de vente de miel de Manuka [2] est une réussite centrale dans ses différentes activités.

Sachant que je suis en pleine reconversion professionnelle dans l'informatique de gestion, il m'a contacté afin de développer une application mobile pour la gestion d'animaux de compagnie.

Son entreprise dispose de compétences solides dans la création de sites internet à l'aide d'un système de gestion de contenu, soit un Content Management System CMS [3]. Elle est experte dans la mise en place de boutiques en lignes avec Shopify [4]. Cependant, pour une application mobile, d'autres ressources sont nécessaires. C'est ainsi que nous nous sommes rencontrés avec son assistant afin de définir ce qu'il était possible d'entreprendre.

MVP Minimum Viable Product

Lors de notre rencontre, ces derniers m'ont présenté un dossier portant le titre de « Minimum Viable Product MVP ». Le premier lien internet que j'ai consulté propose cette définition :

« Le MVP, ou le Minimum Viable Product (Produit Minimum Viable) est une méthode qui a pour objectif de sortir d'abord un produit avec uniquement la fonction la plus attendue (la Killer Feature ou la fonction qui tue) par un public cible, et de proposer le plus rapidement un produit afin de le confronter au marché. Une fois que le produit est confronté au marché, vous pouvez améliorer et enrichir le produit via les méthodes de développement agile. » [5]

La notion de rapidité m'a motivé, voulant développer rapidement mes expériences en programmation. En effet, celles-ci résident dans une formation de deux mois en ligne sur le langage Python, une autre d'un mois sur le langage JavaScript, puis nos cours à l'université. J'ai aussi développé quelques petits projets personnels, comme un site web où il était possible de jouer à un Tetris et un Snake (jeux que j'ai recodé entièrement), puis d'envoyer son score et consulter le classement grâce à un serveur SpringBoot et une base de données MySQL.

De cette manière, je me suis dit que ce serait l'occasion parfaite pour apprendre une nouvelle technologie, et surtout pour développer un projet plus conséquent. Voici le résumé de ce qui m'a été demandé :

1. Une application du nom de « PetZone » permettrait à un utilisateur promenant son animal de consulter son téléphone mobile pour y afficher une carte. Celle-ci présenterait des lieux intéressants tels qu'un point d'eau ou encore une poubelle à sac de déjection. L'application localiserait l'utilisateur et calculerait la distance entre ce dernier et les points affichés.
2. L'utilisateur serait en mesure d'ajouter à la carte de nouveaux points et leurs détails, comme un salon de toilettage ou encore un vétérinaire. Ces points seraient visibles par les autres utilisateurs.
3. L'application mobile permettrait d'alerter les utilisateurs lorsqu'un point ajouté sur la carte consiste en un danger pour les animaux, par exemple une zone d'empoisonnement. Il serait possible de consulter une liste d'alertes environnantes. Un système de notification push informerait l'utilisateur lorsqu'un nouveau danger est ajouté ou alors dès qu'il entre dans une zone qui comprendrait des alertes.
4. L'utilisateur pourrait créer son propre profil personnalisé et celui de ses animaux de compagnie. De cette manière, un système d'authentification sécurisé serait mis en place avec la possibilité de se connecter avec ses comptes issus des réseaux sociaux ou encore de créer une adresse électronique uniquement pour l'application.
5. Les administrateurs détiendraient des droits spécifiques pour surveiller et monitorer l'utilisation de l'application ainsi que les données des utilisateurs. Ils seraient en mesure de gérer des éléments promotionnels.

Pour finir, le dossier qui m'a été transmis décrivait toutes les fonctionnalités d'une application mobile moderne telles qu'une accessibilité aux personnes malvoyantes, une interface adaptable avec des commandes vocales, un référencement maximisé par les moteurs de recherche, un

système de paiement multidevises ou encore l'intégration d'outils d'analyse. L'application serait aussi utilisable à travers un site internet sur un ordinateur.

Il n'est donc pas difficile de comprendre que ce projet dépasse l'ensemble de mes compétences en développement logiciel. Toutefois, nous avons pu en discuter et conscients de ce point, nous avons souhaité aller de l'avant en développant un prototype.

Progressive web application PWA

Lors de notre rencontre, nous avons rapidement abordé les sujets du budget et du choix des technologies. Lorsque le projet m'a été présenté, il m'a été dit que nous n'avions pas de financement pour réaliser l'application. Ensuite, le but était d'éviter les différents magasins de distribution d'Apple et Android car des taxes importantes sont prélevées sur les gains réalisés par l'application.

Nous nous sommes ainsi orientés vers une application web progressive PWA développée grâce au framework Angular et hébergée gratuitement auprès de Firebase. Concernant la base de données, nous avons décidé d'utiliser Firestore qui est un service gratuit inclus dans Firebase. Ces technologies seront développées dans quatrième chapitre de ce travail.

WordPress et communication

Avant de conclure sur l'origine de ce projet, il me semble important de relever ici qu'un site internet WordPress sera réalisé par l'entreprise E-ProShop Sàrl. Voulant profiter des compétences de la société dans ce domaine, nous avons convenu que ce site internet sera créé selon ses habitudes et ses pratiques.

Ce support comprendra par exemple les conditions d'utilisation de l'application. De plus, il intégrera un système de paiement et servira pour développer une stratégie marketing comme celles déjà pratiquées pour les autres boutiques en ligne de la société. C'est donc un point important dont j'ai la chance de ne pas avoir besoin de me soucier pour le développement du prototype.

2.1.2 Nom de l'application

Le dossier qui m'a été présenté mentionne une application du nom de « PetZone ». Cependant, l'entreprise E-ProShop Sàrl a approfondi son analyse de marché et elle s'est rendu compte que celui-ci était déjà pris. C'est donc « PawAtlas » qui a été choisi pour ce travail.

2.1.3 Résumé des fonctionnalités retenues

Les fonctionnalités retenues pour ce prototype ne consistent pas en l'analyse d'applications existantes, mais résident dans des choix parmi les nombreuses possibilités offertes dans le cahier de charges transmis par E-ProShop Sàrl. L'objectif à long terme sera de continuer le projet après mes études afin de répondre à l'ensemble des souhaits de la société.

Par ailleurs, les fonctionnalités implémentées ont fait l'objet de discussions avec le Prof. Dr. Jacques Pasquier-Rocha qui a su m'orienter vers un travail réalisable dans le cadre de mon Bachelor.

Voici brièvement ce qui a été convenu :

1. Une application web progressive PWA sera créée et déployée.
2. Un système d'inscription et de connexion permettra à l'utilisateur d'accéder à l'application.
3. Une carte interactive pourra être consultée et il sera possible d'y inscrire des lieux.
4. L'utilisateur pourra consulter les lieux, notamment ceux qui concernent des dangers.

2.2 Le prototype à réaliser

Nos entretiens avec le Prof. Dr. Jacques Pasquier-Rocha ont permis d'approfondir ce qui était à réaliser pour ce prototype. La section suivante regroupe les cas d'utilisation de l'application puis la modélisation de la base de données.

2.2.1 Cas d'utilisation du Client

Voici le diagramme des cas d'utilisation de l'application d'un point de vue utilisateur. Celui-ci reflète correctement les implémentations effectuées pour le prototype, mais relevons que des ajouts ont été apportés durant la réalisation.

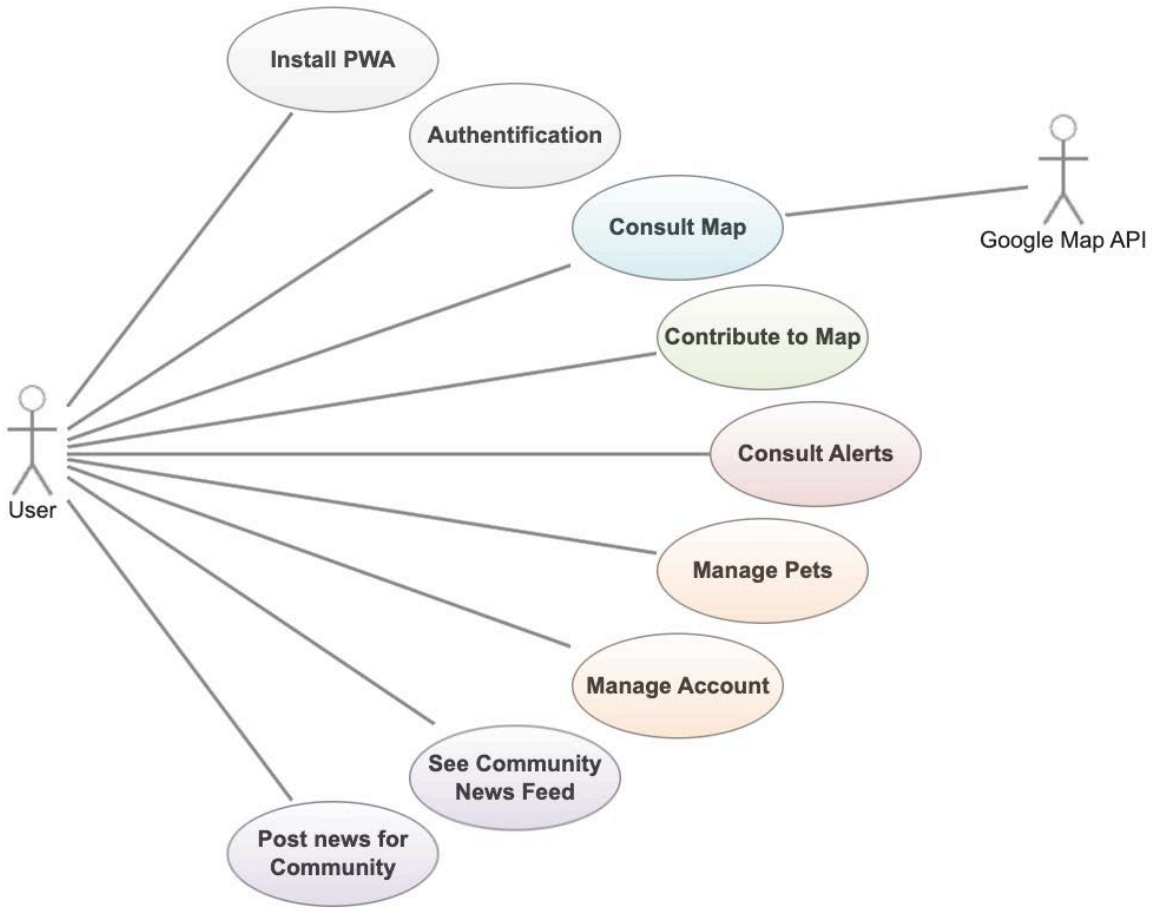

Figure 1 : Diagramme UML des cas d'utilisation

Install PWA

Le premier cas d'utilisation permet à l'utilisateur d'installer l'application web progressive PWA. Nous verrons dans le chapitre suivant que ce processus n'est pas automatique et ne se fait pas à travers les magasins de distribution usuels.

Authentification

Ce deuxième cas d'utilisation comporte tout le processus d'authentification à l'application. Pour des raisons de simplicité et de clarté, le diagramme présenté ne précise pas le détail de ce processus. Relevons que l'utilisateur peut soit créer un compte, soit utiliser un compte issu de ses réseaux sociaux. Le système d'authentification sécurisé se fait grâce au service Firebase Authentification que nous expliquerons dans le chapitre dédié à l'implémentation.

Consult Map

L'utilisateur peut consulter une carte qui provient de l'API Google Map. Nous verrons dans les choix d'implémentations que pour des raisons budgétaires, le prototype a été réalisé avec l'API OpenStreetMap.

De plus, l'utilisateur est en mesure de :

- Filtrer les lieux indiqués (ses propres contributions ou toutes).
- Filtrer le type de lieux affichés (les alertes ou les intérêts, mais aussi le type de lieux).
- Afficher le détail d'un lieu.
- Zoomer ou agrandir la zone affichée sur la carte.
- Se localiser et ajuster la carte sur sa position.

Contribute to Map

Ce cas d'utilisation précise que l'utilisateur peut contribuer à la carte. Encore une fois, le diagramme ne détaille pas l'ensemble des processus, mais ce point inclut les différentes possibilités suivantes :

- Ajouter un lieu.
- Modifier un lieu.
- Supprimer un lieu.

Consult Alerts

Ici, l'utilisateur peut consulter la liste des alertes ajoutées sur la carte, notamment :

- Consulter et trier la liste des alertes.
- Modifier la localisation et le rayon des alertes présentes dans la liste.
- Afficher le détail des alertes.

Autres cas d'utilisation

Les autres cas d'utilisation présents dans le diagramme ne sont pas détaillés ici car ils n'ont pas été implémentés dans le prototype réalisé dans le cadre de ce travail de Bachelor. Ceux-ci comprennent des fonctionnalités liées à la gestion de profils personnalisés et à un aspect communautaire où il est possible de consulter ou d'enrichir un fil d'actualités.

2.2.2 Cas d'utilisation des administrateurs

Au niveau du code, le prototype implémenté dans le cadre de ce travail de Bachelor ne comprend pas de différenciation entre un utilisateur et un administrateur de l'application. Cependant, les fonctionnalités d'administration sont présentes dans la console Firebase [6] où l'application a été déployée. Un administrateur qui en a l'accès peut se connecter depuis n'importe quel navigateur internet pour gérer l'application. Ci-dessous, voici un aperçu des cas d'utilisation disponibles d'un point de vue administrateur. Il y en a quantité d'autres, c'est pourquoi nous n'avons pas réalisé de diagramme en comparaison à la section précédente.

Aperçu

Dès qu'un administrateur se connecte dans la console Firebase, il a accès à une vue d'ensemble du projet. Celle-ci permet de consulter le nombre de téléchargements de l'application, soit le nombre de fois où un utilisateur s'est connecté à son l'URL pour obtenir son contenu html, les feuilles de style css et le code JavaScript. Il peut aussi y voir le nombre de lectures et d'écritures réalisées sur la base de données ou encore l'état de son espace de stockage.

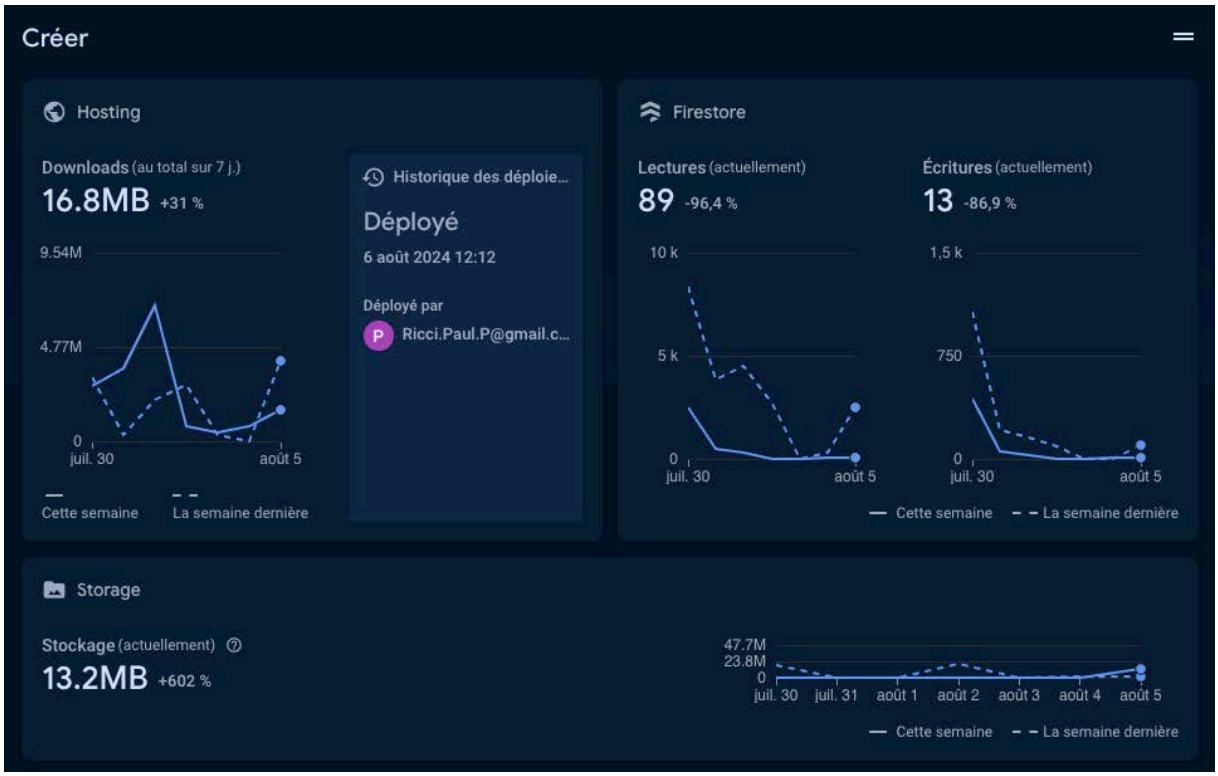

Figure 2 : Aperçu de l'application dans la console Firebase

Hébergement

L'administrateur a aussi accès aux différents éléments qui concernent l'hébergement de l'application. Ceux-ci regroupent la version actuelle de l'application ainsi que l'adresse électronique de la personne qui l'a déployée. Il peut y voir les noms de domaines attribués automatiquement par Firebase. Relevons que ceux-ci sont sécurisés et disposent gratuitement d'un certificat SSL. Il est possible de rajouter un nom de domaine personnalisé.

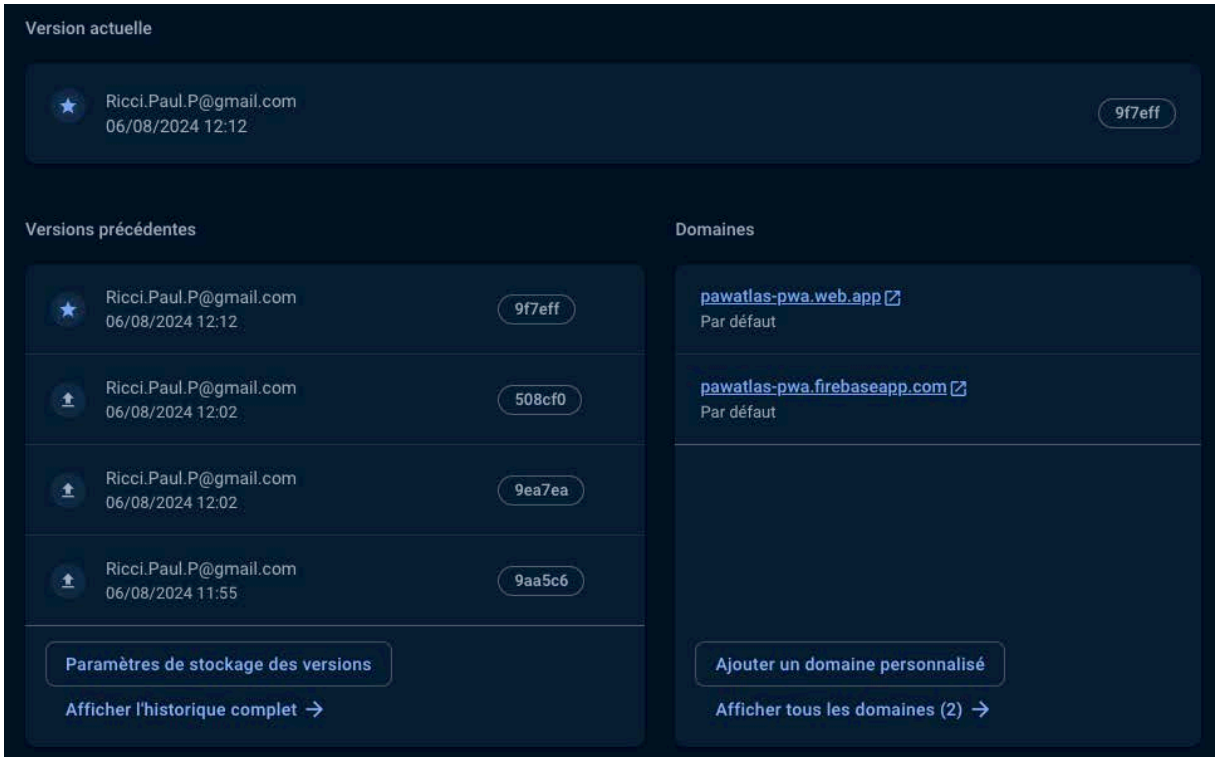

Figure 3 : Hébergement de l'application dans la console Firebase

Authentification

Il est possible de configurer différentes méthodes de connexion comme celles avec les réseaux sociaux. Des modèles de courriel sont disponibles tels que ceux envoyés à un utilisateur ayant oublié son mot de passe. La liste des utilisateurs inscrits est visible. Depuis celle-ci, l'administrateur est en mesure de réinitialiser le mot de passe d'un compte, le désactiver ou encore le supprimer.

Figure 4 : Liste des utilisateurs authentifiés dans la console Firebase

Base de données

La base de données est hébergée grâce au service Firestore. Un administrateur y a accès depuis la console Firebase. Il peut y effectuer des actions classiques comme ajouter, supprimer ou modifier des données. Des collections peuvent être créées et tout comme des règles pour autoriser ou non les opérations sur la base de données.

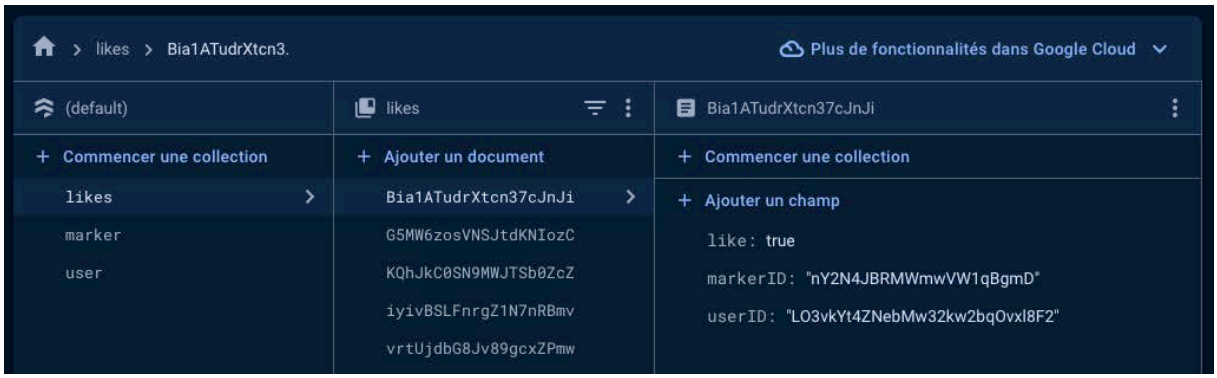

Figure 5 : Base de données Firestore dans la console Firebase

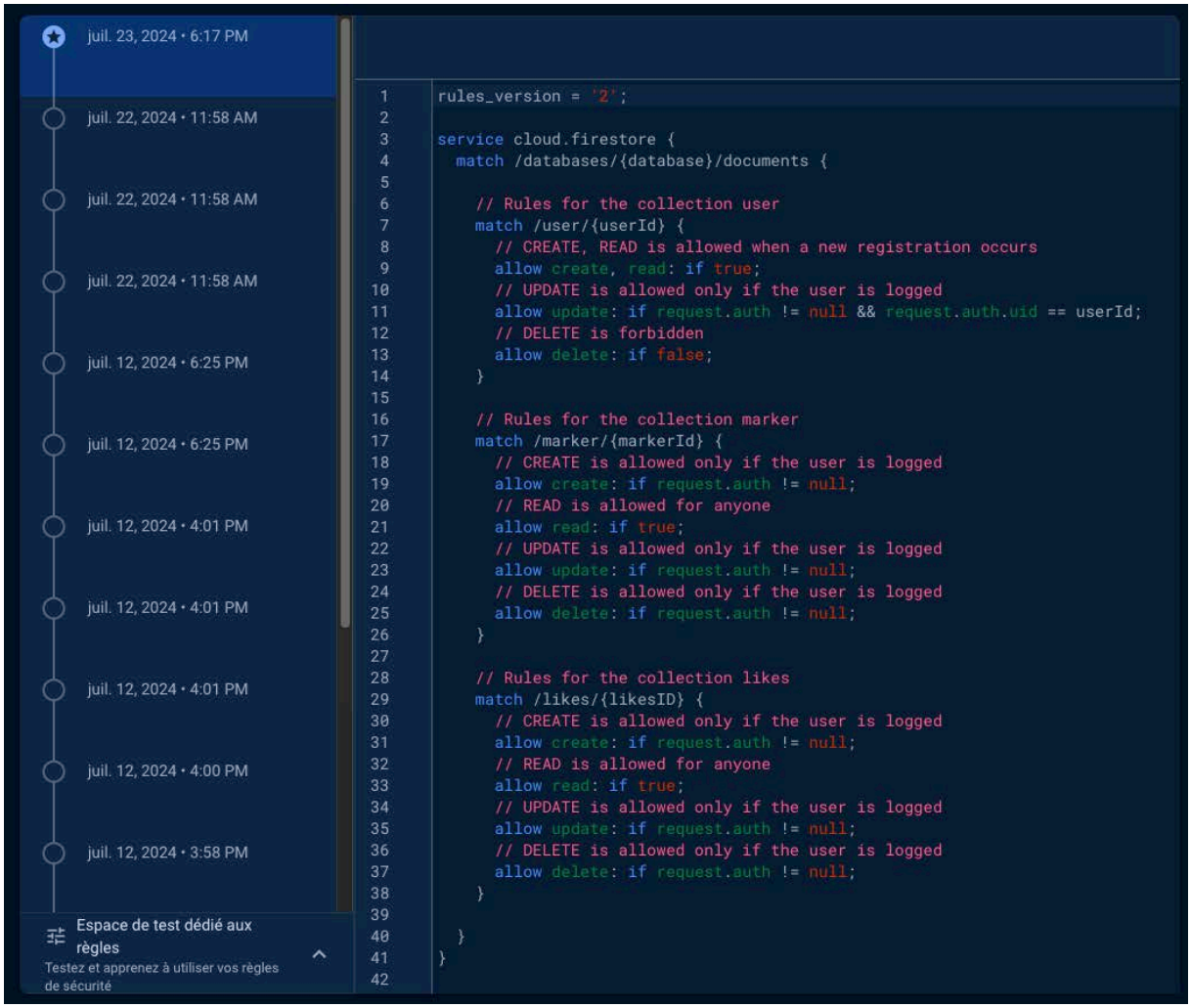

Figure 6 : Règles d'accès à la base de données dans la console Firebase

Espace de stockage

Dans le prototype réalisé, il est possible d'ajouter des lieux sur la carte interactive et d'y joindre une image. Les images insérées ne sont pas stockées dans la base de données Firestore mais dans un autre service appelé Storage. Un administrateur a accès aux images. Il peut en ajouter ou en supprimer à partir de la console Firebase.

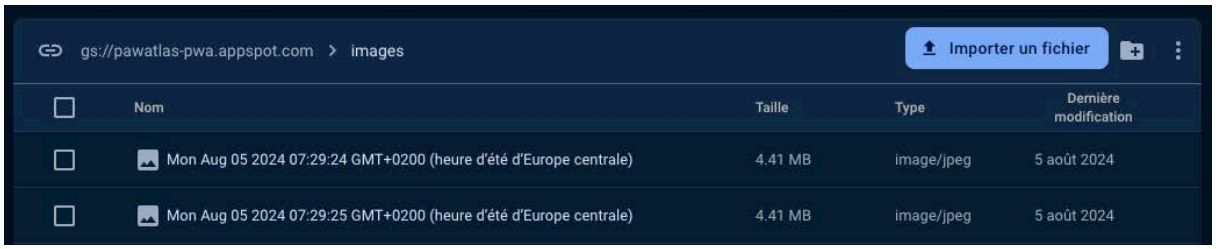

Figure 7 : Espace de stockage dans la console Firebase

Utilisation et Facturation

Pour finir, un administrateur qui a accès à la console Firebase est en mesure de consulter de manière détaillée les données d'utilisation et de facturation de l'application.

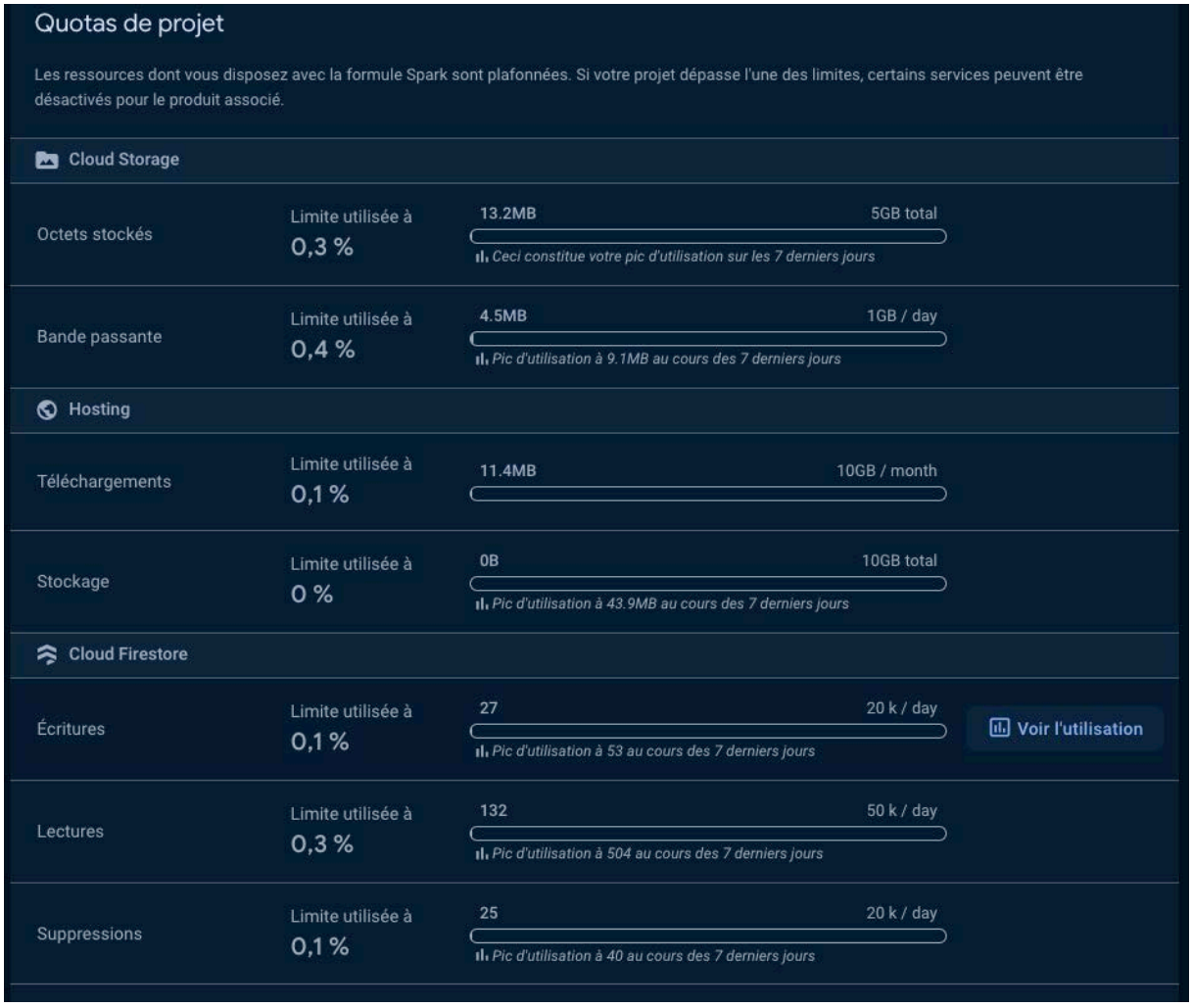

Figure 8 : Utilisation et facturation du projet dans la console Firebase

2.2.3 Modélisation de la base de données

Modéliser une base de données est une étape importante dans la réalisation d'un logiciel. Elle permet de définir les données nécessaires à l'implémentation d'une application. Les termes employés par le Dr. Lena Wiese résument extrêmement bien ce processus :

« Database design is a phase before using a database system - even before deciding which system to use. The design phase should clearly answer basic questions like: Which data are relevant for the customers or the external applications? How should these relevant data be stored in the database? » [7]

Dans le prototype d'application réalisé pour ce travail de Bachelor, notre choix s'est porté pour des raisons financières vers les services de Firebase qui sont gratuits jusqu'à un certain stade d'utilisation. Ce choix n'est pas sans conséquences car la base de données Firestore est de type NoSQL, ce qui laisse une grande liberté concernant les données.

Toutefois, une phase de modélisation reste primordiale à nos yeux. Un diagramme est utile car c'est au niveau du code qu'il est nécessaire de bien réfléchir quant aux types de données que nous allons manipuler, à leur structure et à leurs relations. À ce sujet, Rudi Bruchez affirme que les bases de données NoSQL proposent :

« une approche dite « sans schéma », ou à schéma « relaxé », ce qui veut dire qu'aucune vérification ou contrainte de schéma n'est effectuée par le moteur. C'est au développeur qui utilise ces systèmes de décider comment il organise ses données, et ceci dans son code client. » [8]

Ce point n'est pas anodin. Par exemple, il n'est pas possible de configurer des règles de suppression et de modification des données en cascade dans une base de données de type NoSQL. Cela se fait directement dans le code, d'où l'importance d'avoir en tête la structure de nos données et leurs relations.

Nous pouvons citer encore une fois les termes du Dr. Lena Wiese, qui affirme que la phase de modélisation est tout aussi essentielle lorsque la base de données n'impose pas de schéma :

« For conventional database systems (with a more or less fixed data schema) changing the schema on a running database system is complex and costly; that is why a good database design is essential for these systems. Nevertheless, for database systems with more flexible schemas (or no schema at all), the design phase is important, too: identifying relationships in the data, grouping data that are often accessed together, or choosing good values for row keys or column names are all beneficial for a good performance of the database system. » [7]

De cette manière, nous avons réalisé un diagramme Entité-Relation pour modéliser les données nécessaires à notre prototype. Ce type de modèle a été proposé en 1976 par Peter Chen dans un article publié au journal ACM Transactions on Database Systems qui relève son efficacité à représenter les objets du monde réel :

« The entity-relationship model adopts the more natural view that the real world consists of entities and relationships. It incorporates some of the important semantic information about the real world. » [9]

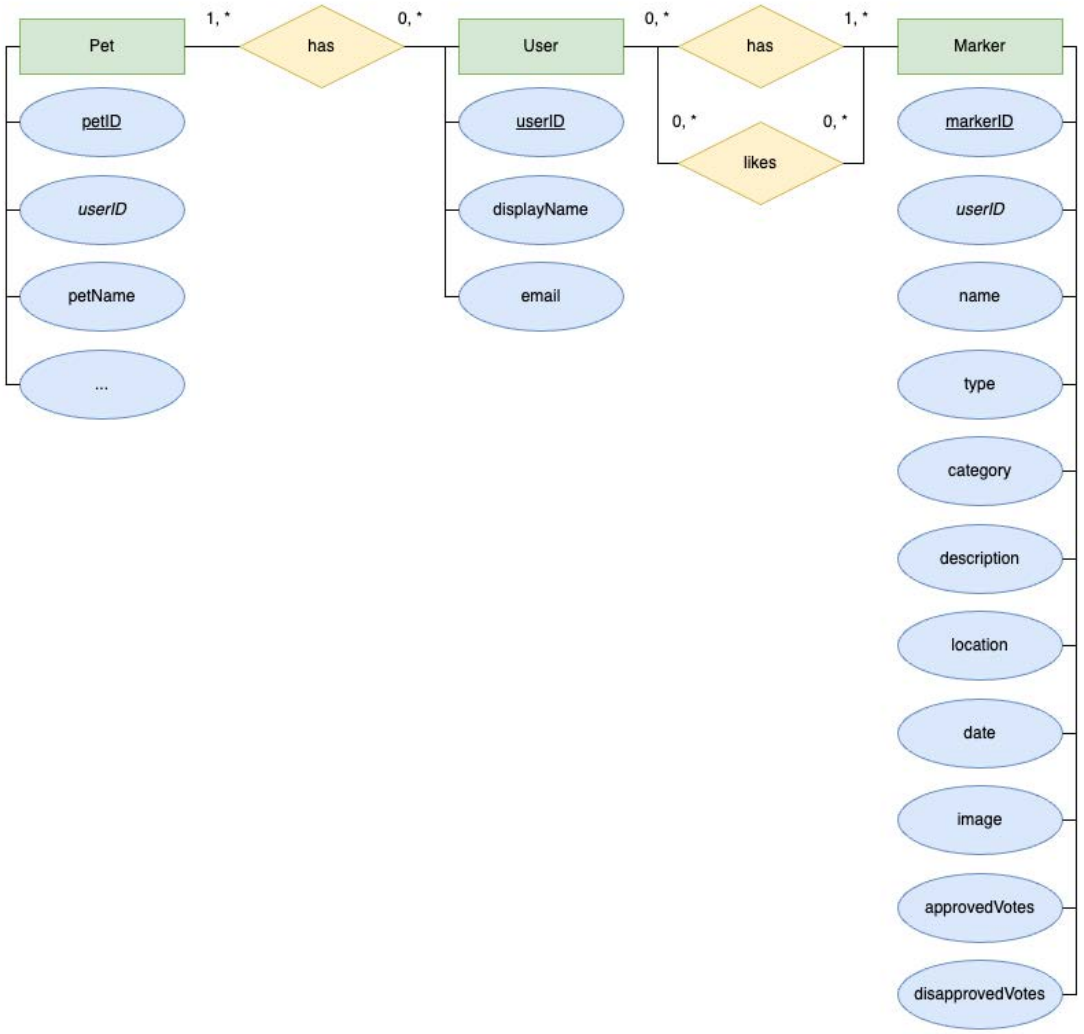

Figure 9 : Diagramme ERM de la base de données

Ce diagramme comporte trois entités principales :

1. Pet, soit les animaux de compagnie des utilisateurs.
2. User, soit les utilisateurs.
3. Marker, soit les lieux affichés sur la carte interactive.

Dans notre prototype, les cas d'utilisation liés à la création de profils pour les animaux de compagnie n'ont pas été implémentés. La modélisation de l'entité Pet n'est donc pas terminée.

Pour notre diagramme, nous avons ajusté les cardinalités en remplaçant la notation de Chen par la notation appelée « min-max ». Celle-ci permet de définir le nombre minimum et maximum d'entités qui sont engagées dans une relation. Par exemple, nous pouvons voir que *« 0 ou plusieurs User sont en relation avec Marker, et que 1 ou plusieurs Marker sont en relation avec User ».*

L'entité User comporte un userID qui permet d'identifier de manière unique un utilisateur. Cet identifiant sert de clé étrangère pour l'entité Marker car en effet, un lieu ajouté sur la carte interactive appartient à un utilisateur précis. Il ne peut être modifié ou supprimé par quelqu'un d'autre.

Un cas d'utilisation supplémentaire a été implémenté dans notre application. Un utilisateur peut approuver ou rejeter un lieu ajouté sur la carte. C'est pourquoi la relation Likes a été modélisée et que des attributs liés à ces votes ont été ajoutés à l'entité Marker. Cette relation permet de stocker de manière précise quel lieu a été apprécié ou non par un utilisateur et de retrouver facilement cette information dans la base de données.

Pour terminer l'entité Marker comporte un attribut « type ». Ce dernier a été définis à partir d'une liste de catégories fournie par E-ProShop Sàrl que nous avons sauvegardée dans un fichier TypeScript dans notre code :

```
1     export const interestCategories = [
2       { value: 'SPA', label: 'SPA' },
3       { value: 'Veterinaire', label: 'Vétérinaire' },
4       { value: 'Toilettage', label: 'Toilettage' },
5       { value: 'Animalerie', label: 'Animalerie' },
6       { value: 'Medecine', label: 'Médecine alternative' },
7       { value: 'Poubelle', label: 'Poubelle avec sacs' },
8       { value: 'Eau', label: "Point d'eau" },
9       { value: 'Parc', label: 'Parc' },
10      { value: 'Centre', label: 'Centre animalier' },
11      { value: 'Garde', label: "Garde d'animaux" },
12      { value: 'Baignade', label: 'Zone de baignade' },
13      { value: 'Loisirs', label: 'Centre de loisirs' },
14      { value: 'Autre intérêt', label: 'Autre intérêt' },
15    ];
16
17    export const dangerCategories = [
18      { value: 'Toxique', label: 'Toxique' },
19      { value: 'Patous', label: 'Présence de patous' },
20      { value: 'Cyanobactéries', label: 'Présence de cyanobactéries'
      },
21      { value: 'Rage', label: 'Présence de rage' },
22      { value: 'Chasse', label: 'Zone de chasse' },
23      { value: 'Bétail', label: 'Présence de bétails' },
24      { value: 'Interdit', label: 'Zone interdite aux animaux' },
25      { value: 'Autre danger', label: 'Autre danger' },
26    ];
```

Code 2 : Interface définissant les catégories de l'entité Marker

Un Marker contient différentes catégories appartenant à deux types, un intérêt ou alors un danger. Il aurait été préférable de modéliser ceci en créant deux entités distinctes qui seraient une spécialisation de Marker, comme représenté dans la figure 10 qui suit.

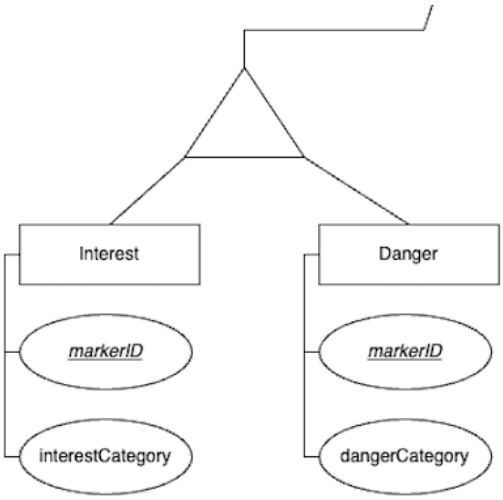

Figure 10 : Spécialisation de Marker en deux types

Ceci serait optimal dans une base de données relationnelle où les jointures sont facilement réalisables. Dans notre implémentation, cela augmenterait le nombre de requêtes à notre base de données NoSQL. L'ensemble des informations se trouverait dans deux collections différentes qu'il faudrait interroger et la jointure se ferait dans le code.

En effet, chaque requête est facturée dans l'environnement Firebase. C'est pourquoi cette spécialisation n'a pas été modélisée dans le diagramme que nous venons de parcourir.

3

Point de vue utilisateur

3.1 Le prototype réalisé

Il y a quelques différences entre ce qui a été effectivement implémenté et ce qui avait été convenu avec le Prof. Dr. Jacques Pasquier-Rocha. Les cas d'utilisation présentés précédemment ont été respectés, mais le prototype réalisé comporte quelques ajouts supplémentaires. Dans ce chapitre, nous allons parcourir en détail les différentes fonctionnalités s'offrant à l'utilisateur de notre application.

3.1.1 Le design

Il nous semble important de préciser que le design de l'application n'est pas définitif. Celui-ci a été réalisé de manière rapide en utilisant des composants Bootstrap [10] prêts à l'emploi. La bibliothèque Fontawesome [11] a permis d'obtenir gratuitement des icônes. Le logo est celui définit par défaut dans Angular lors de la création d'une application web progressive. Pour finir, les images utilisées dans cette présentation sont libres de droits [12].

3.2 Installation de la PWA

L'installation d'une application web progressive ne se fait pas à travers les magasins de distributions traditionnels et diffère selon le type de périphérique. Lorsqu'un utilisateur veut accéder à l'application, il doit d'abord se rendre à l'adresse suivante à l'aide d'un navigateur internet : https://pawatlas-pwa.web.app/

L'utilisateur verra ensuite un message lui proposant l'installation de l'application. Par ailleurs, si ce dernier est déjà connecté, il sera redirigé automatiquement à la page « /map », et inversement à la page « /login ».

3.2.1 Installation sur un ordinateur ou sur Android

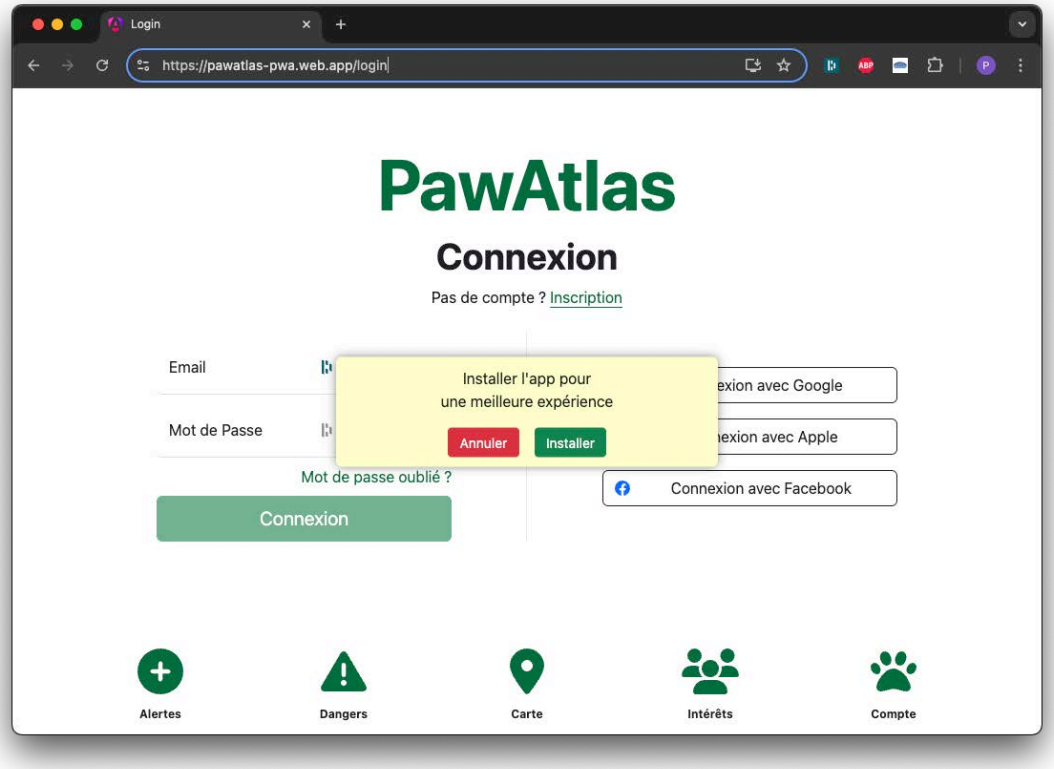

Figure 11 : Page d'accueil de l'application sur un MacBook dans Chrome

Sur ordinateur, un popup apparait proposant d'installer l'application pour une meilleure expérience. C'est ici que réside la force des application web progressive : Ce sont en réalité un site internet qu'il est possible d'installer sur son périphérique.

Dorénavant, plusieurs choix sont possibles. Le premier serait de cliquer sur « Annuler », ce qui fermerait simplement le popup mais n'entraverait pas l'accès à l'application. Le second serait d'installer la PWA. Le dernier consisterait à ne pas lancer l'installation à l'aide du bouton « Installer », mais déclencher ce processus en cliquant sur la première icône représentant un ordinateur et une flèche dans la barre d'URL.

Le processus d'installation démarre avec une alerte qui demande à l'utilisateur de confirmer. Un dossier s'ouvre avec l'emplacement où l'application a été installée. Pour la désinstaller, il est nécessaire de l'ouvrir à nouveau et de cliquer sur le menu en haut à droite.

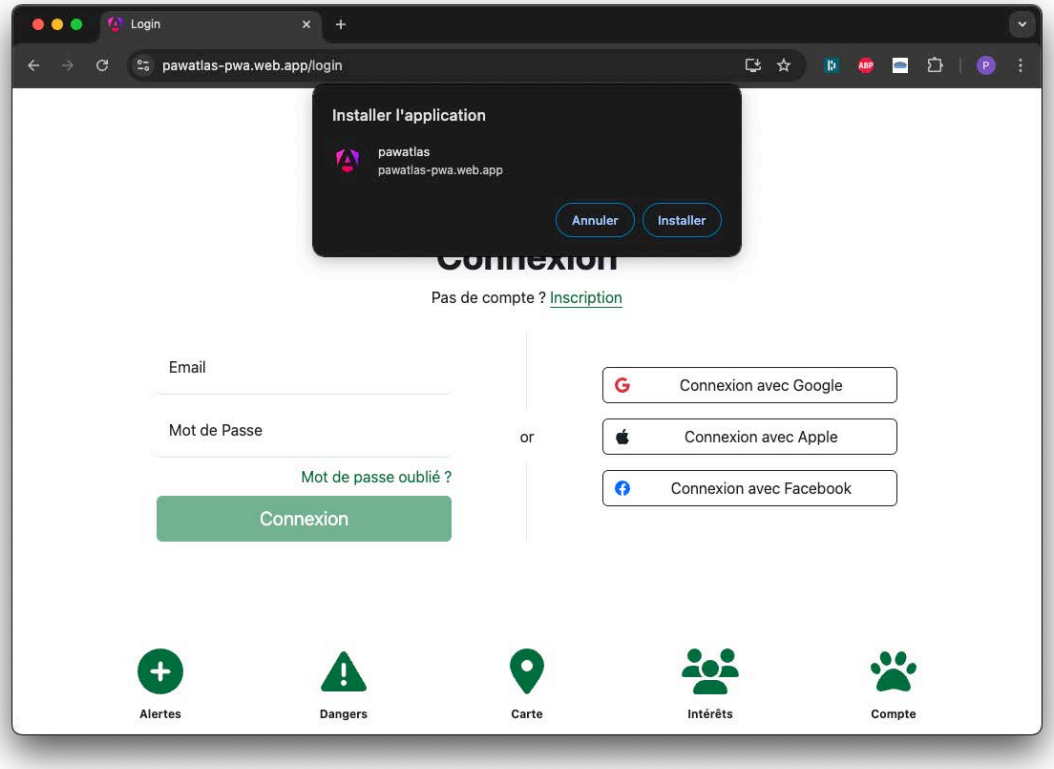

Figure 12 : Alerte de confirmation d'installation sur un MacBook dans Chrome

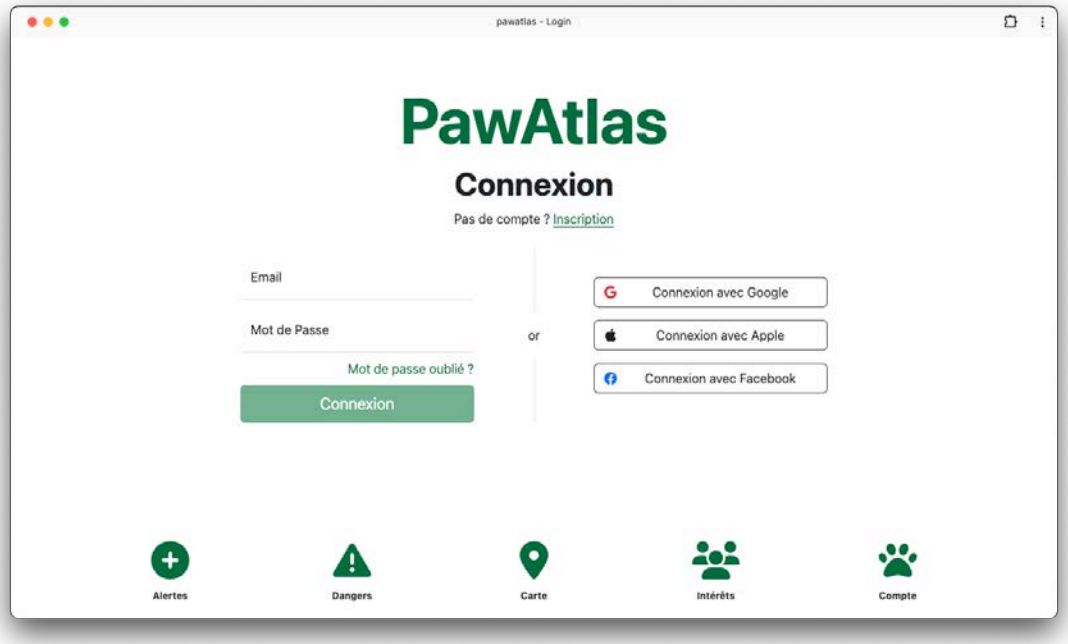

Figure 13 : Application installée sur un MacBook

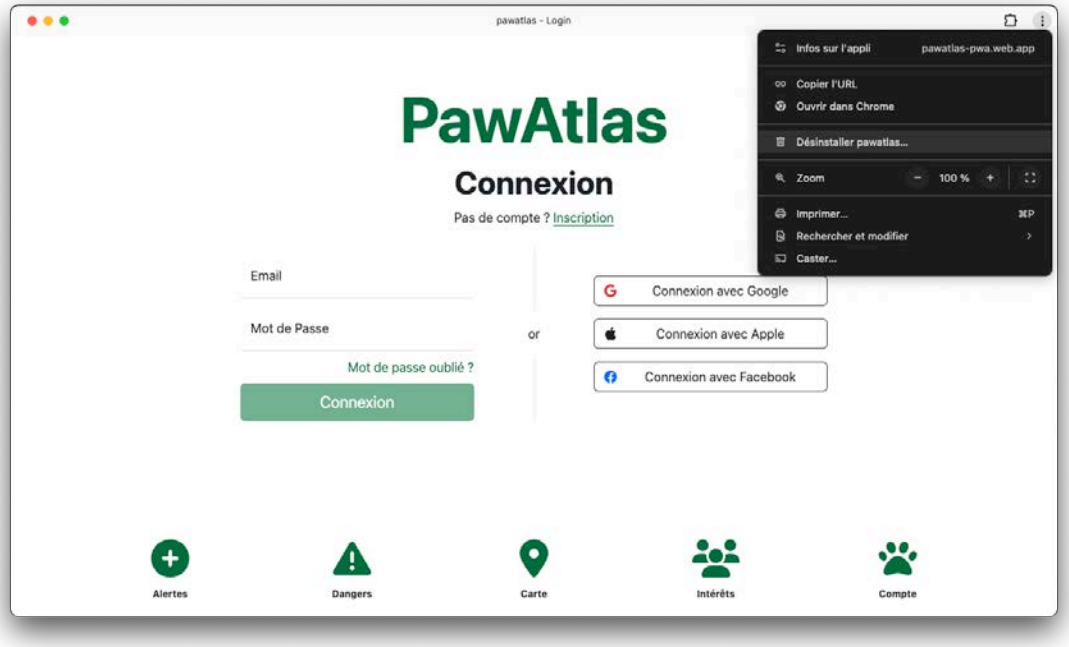

Figure 14 : Désinstallation de l'application avec le menu en haut à droite

Sur un téléphone Android, les étapes sont exactement les mêmes que sur un ordinateur. Cependant, la PWA vient s'installer sur l'écran d'accueil du téléphone et pour la supprimer, il faut le faire de la même manière que pour tout autre application.

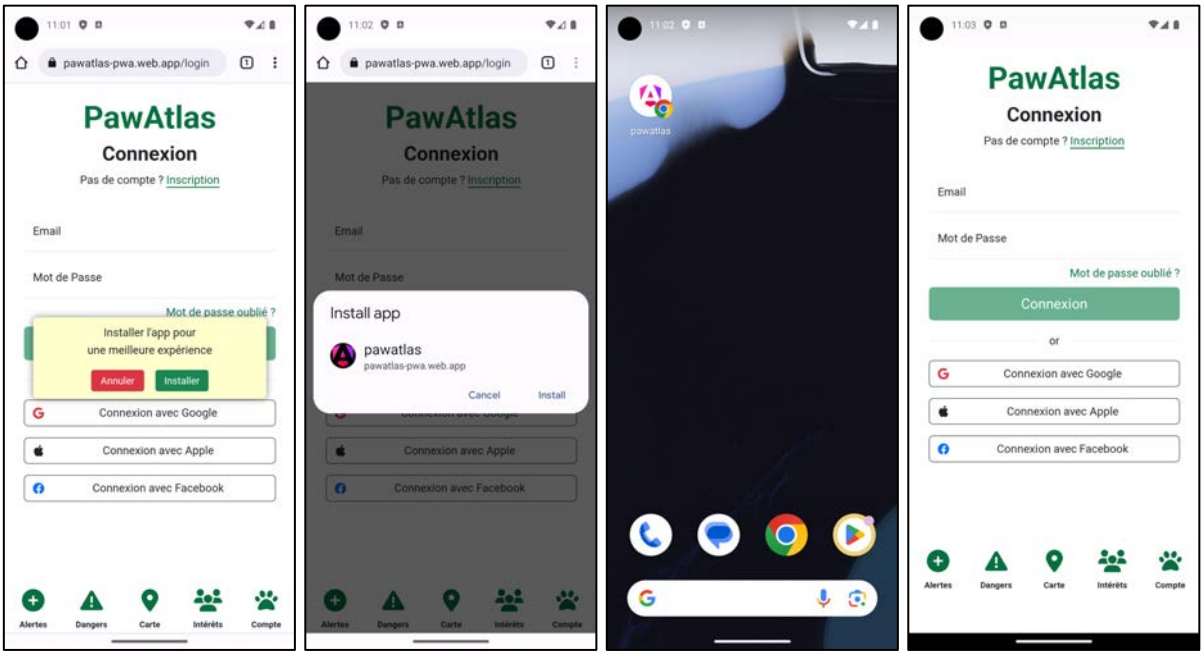

Figure 15 : Processus d'installation sur Android

3.2.2 Installation sur iOS

Concernant les mobile iOS, le processus d'installation est plus délicat. Les application web progressives sont une technologie assez récente et Apple n'a pas encore permis l'installation en un clic depuis un navigateur internet :

« If your application is opened in Android's browser (Chromium-based, like Chrome or Brave), the event beforeinstallprompt will be fired, and the browser may show a prompt to install the app. But when it comes to iOS — it not so obvious. A user has to manually select the Safari's menu item to install the app. And if the user isn't experienced how to do it or does not know that this app could be installed the chances that he will install your app is minimal. » [13]

Il a donc été nécessaire de modifier le message d'installation pour qu'il donne suffisamment d'instructions à l'utilisateur. Supprimer l'application se fait de manière habituelle.

Figure 16 : Message d'installation iOS

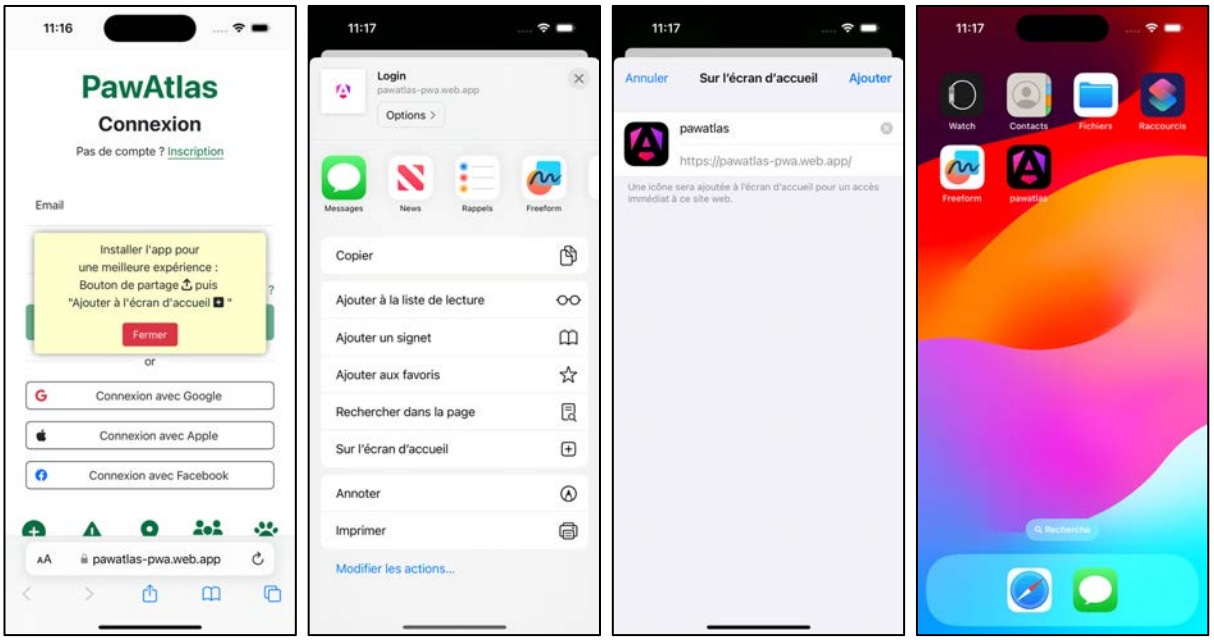

Figure 17 : Processus d'installation sur iOS

3.2.3 Mise à jour

Lorsque l'application est mise à jour, l'utilisateur recevra une alerte pour l'installer.

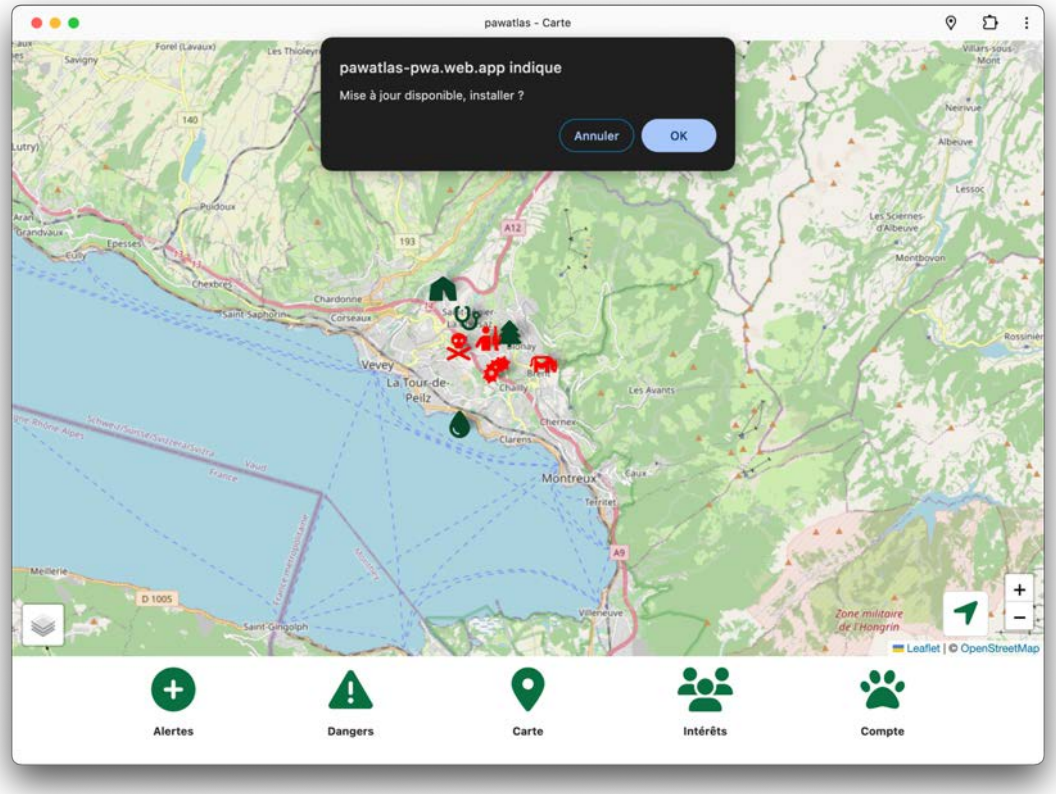

Figure 18 : Proposition de mise à jour

3.3 Connexion

Afin d'alléger ce travail écrit en images, les fonctionnalités présentées à partir de ce stade porteront sur un ordinateur et un appareil mobile iOS. Ce dernier ne diffère que très peu sur Android.

La page de connexion est le point d'entrée de l'application. Elle permet à l'utilisateur de se connecter avec un courriel et un mot de passe. Aussi, il peut le faire à l'aide de ses comptes Google, Apple ou Facebook, mais nous n'avons pas implémenté ces deux dernières possibilités.

En effet, mettre en place un système de connexion avec un compte Apple nécessitait de rejoindre le programme pour développeurs auprès de l'entreprise [14]. Concernant Facebook, cela impliquait d'enregistrer l'application auprès du groupe [15] qui demande ensuite un contrôle périodique des données collectées. Nous n'avons pas entrepris ces démarches pour le prototype.

3.3.1 Connexion avec courriel et mot de passe

La connexion avec un courriel se fait de manière classique à l'aide d'un formulaire. Le bouton « Connexion » n'est pas actif tant que les champs sont vides ou encore si un courriel ne comporte pas une structure adéquate. Lorsque l'utilisateur commet une erreur dans ses identifiants, il est averti par un message.

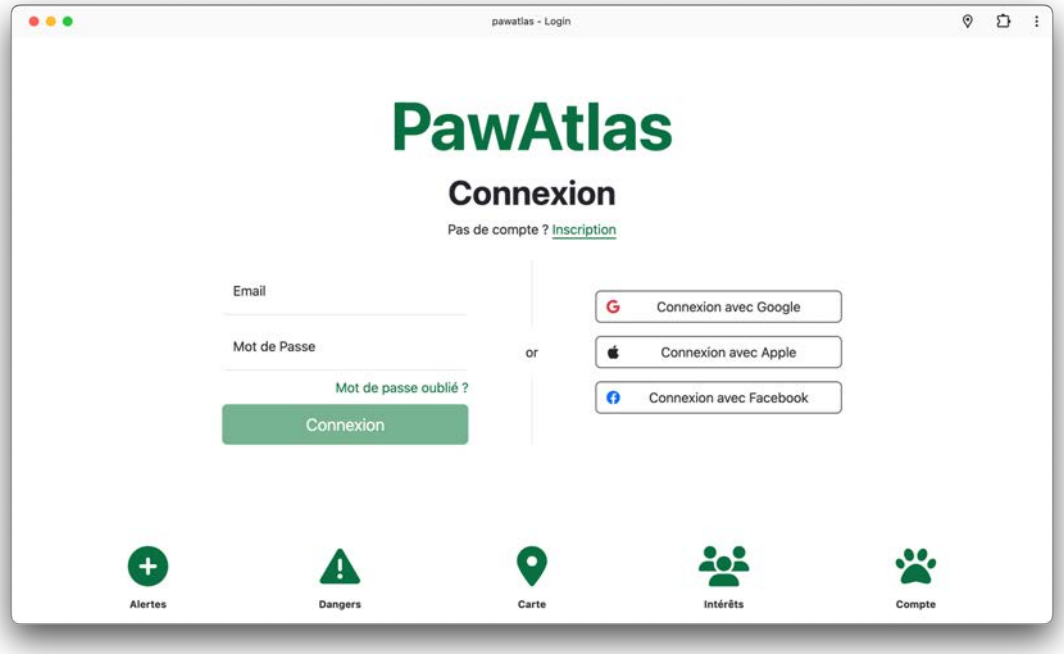

Figure 19 : Connexion avec courriel et mot de passe

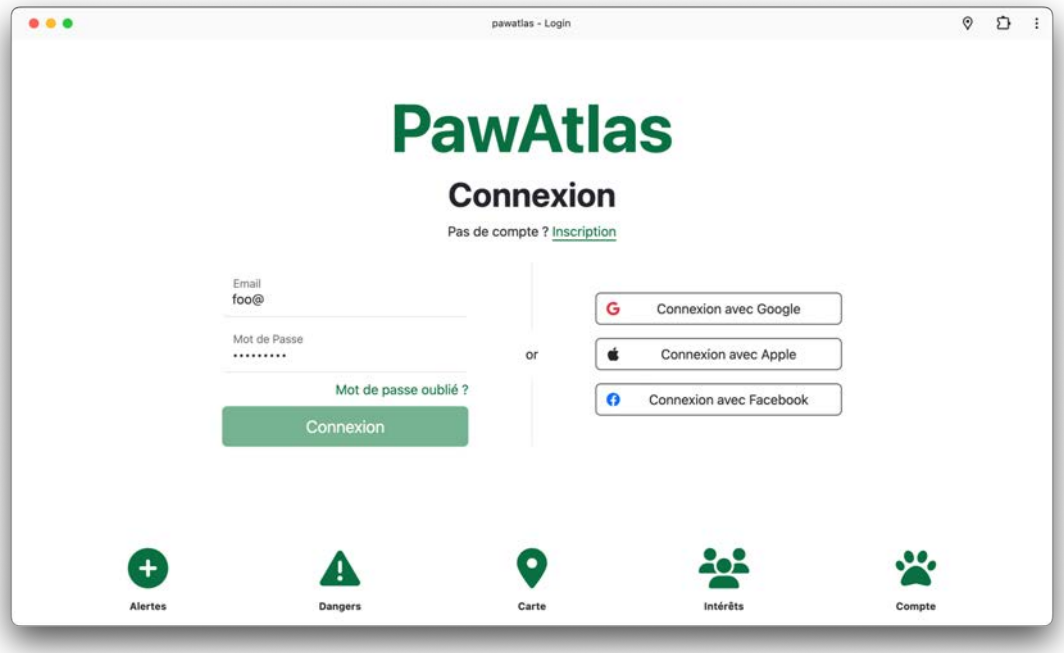

Figure 20 : Connexion avec courriel et mot de passe - bouton inactif

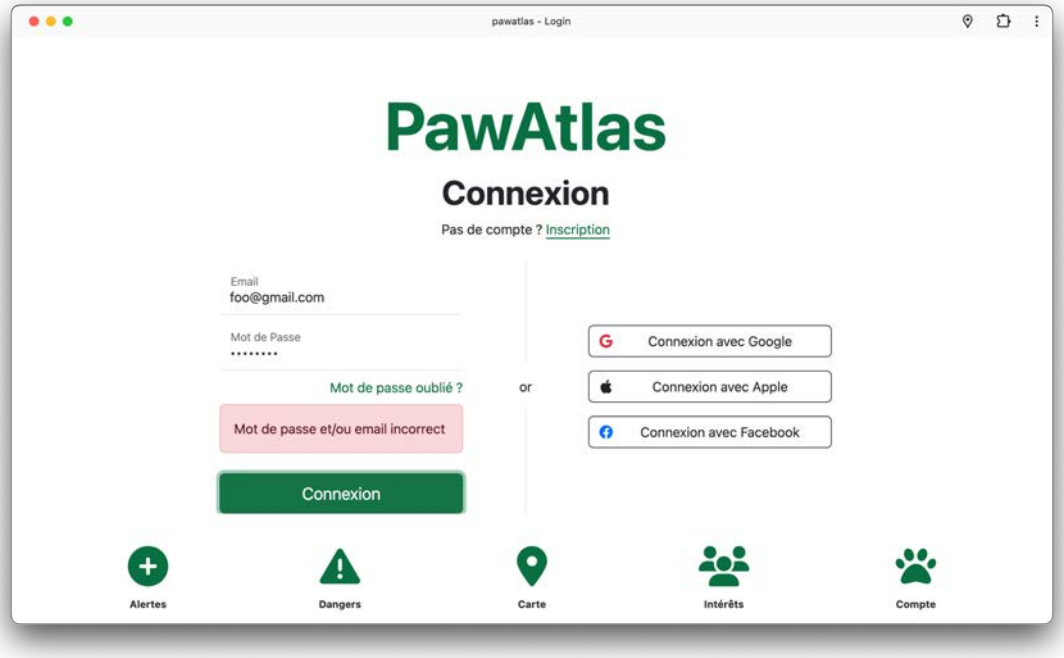

Figure 21 : Connexion avec courriel et mot de passe - message d'erreur

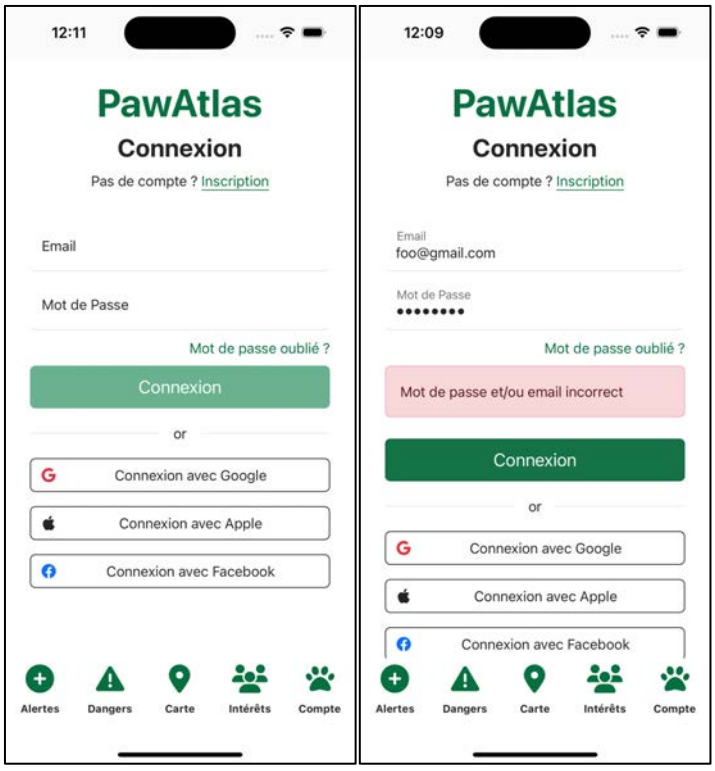

Figure 22 : Connexion avec courriel et mot de passe - version mobile

3.3.2 Connexion avec Google

Quand l'utilisateur clique sur la méthode de connexion avec Google, cela ouvre une fenêtre qui permet de choisir son compte et d'y entrer son mot de passe.

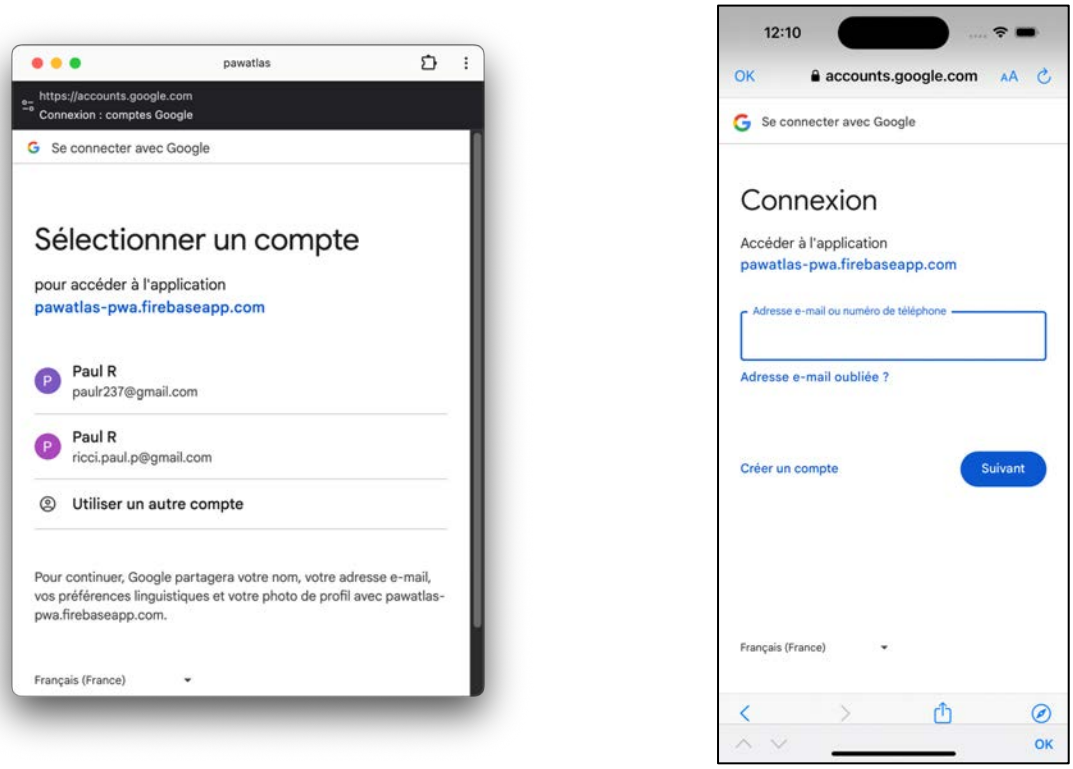

Figure 23 : Connexion avec Google - sur ordinateur et version mobile

3.4 Navigation

À partir de la page de connexion, il est possible de cliquer sur le lien « Inscription » afin d'être redirigé. Un utilisateur qui a oublié son mot de passe peut cliquer sur le lien « Mot de passe oublié ». Cette dernière fonctionnalité n'a pas été implémentée pour le prototype, mais elle est facilement réalisable grâce à la console Firebase où des modèles de courriel sont proposés.

Nous pouvons apercevoir le menu de navigation qui se trouve au bas de l'application. Celui-ci reste visible en permanence. L'accès aux différentes pages est restreint si un utilisateur n'est pas connecté. Appuyer sur ces onglets n'aura aucun effet.

3.5 Inscription

La page d'inscription permet à l'utilisateur de se connecter à l'aide d'un formulaire où il est invité à y insérer un identifiant. Celui-ci servira plus tard lors de la création d'un profil personnalisé qui dépasse les cas d'utilisation de ce prototype. Cependant, cet identifiant est déjà

sauvegardé dans la base de données. Si l'utilisateur s'est connecté avec son compte Google, c'est celui qu'il aura paramétré auprès de ce fournisseur qui est récupéré.

L'utilisateur devra ensuite choisir une adresse électronique et un mot de passe qu'il devra confirmer. À ce stade du prototype, nous n'avons pas encore implémenté la validation de courriel, mais cela reste encore une fois tout à fait possible à l'aide de Firebase. Concernant le mot de passe, celui-ci doit contenir au moins un chiffre, une majuscule et six caractères.

L'utilisateur doit accepter les « conditions d'utilisations ». Ce lien sera par la suite une redirection sur une page dédiée sur le site internet réalisé par l'entreprise E-ProShop Sàrl.

Nous pouvons remarquer dans les images suivantes que tant que l'utilisateur ne remplit pas les critères mentionnés ci-dessus, le bouton « Inscription » est inactif. De plus, des messages d'erreur l'avertissent si un identifiant ou un courriel est déjà utilisé.

L'utilisateur peut ici aussi décider de se connecter à l'aide de son compte Google de la même manière que sur la page de connexion.

Après avoir effectué son inscription, l'utilisateur est automatiquement connecté et redirigé à l'adresse « /map » qui correspond à l'onglet « Carte » dans le menu de navigation.

Relevons que grâce au système d'authentification de Firebase, un utilisateur connecté n'a pas besoin de se reconnecter lors de sa prochaine visite :

« Pour une application Web, le comportement par défaut consiste à conserver la session d'un utilisateur même après que celui-ci ferme le navigateur. Ceci est pratique car l'utilisateur n'est pas obligé de se connecter en permanence à chaque fois que la page Web est visitée sur le même appareil. Cela pourrait obliger l'utilisateur à devoir ressaisir son mot de passe, envoyer une vérification par SMS, etc., ce qui pourrait ajouter beaucoup de friction à l'expérience utilisateur. » [16]

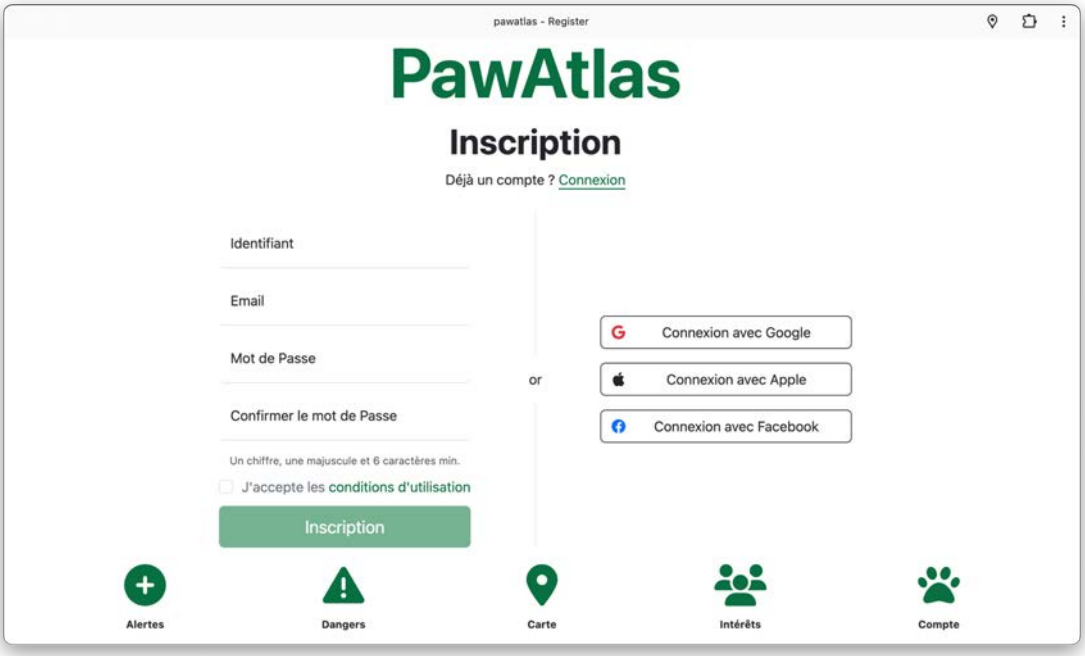

Figure 24 : Inscription

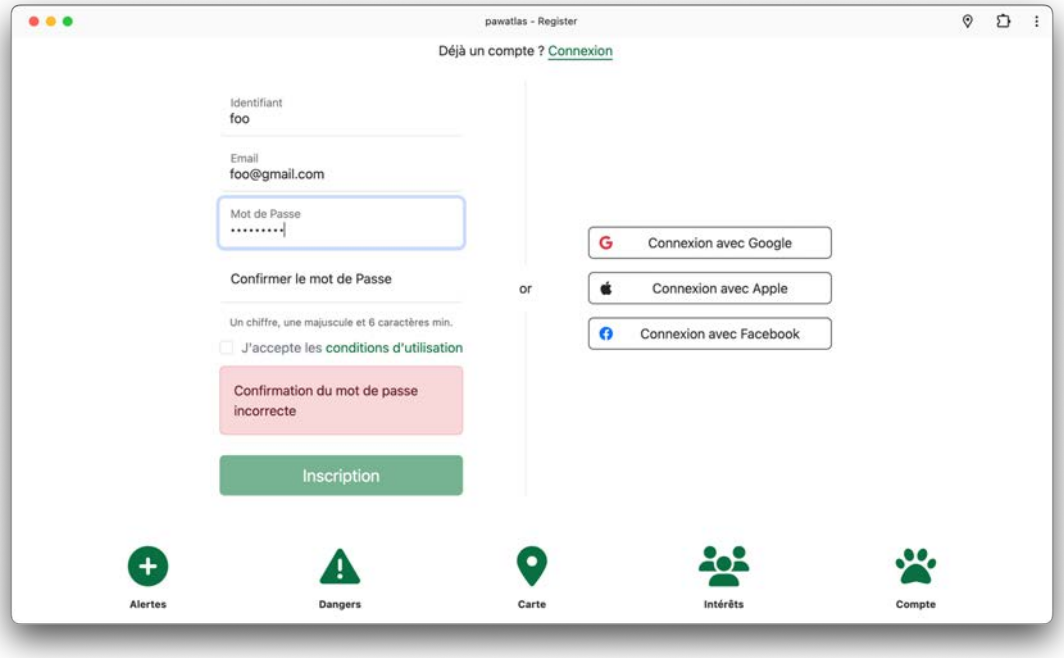

Figure 25 : Inscription - confirmation du mot de passe incorrecte

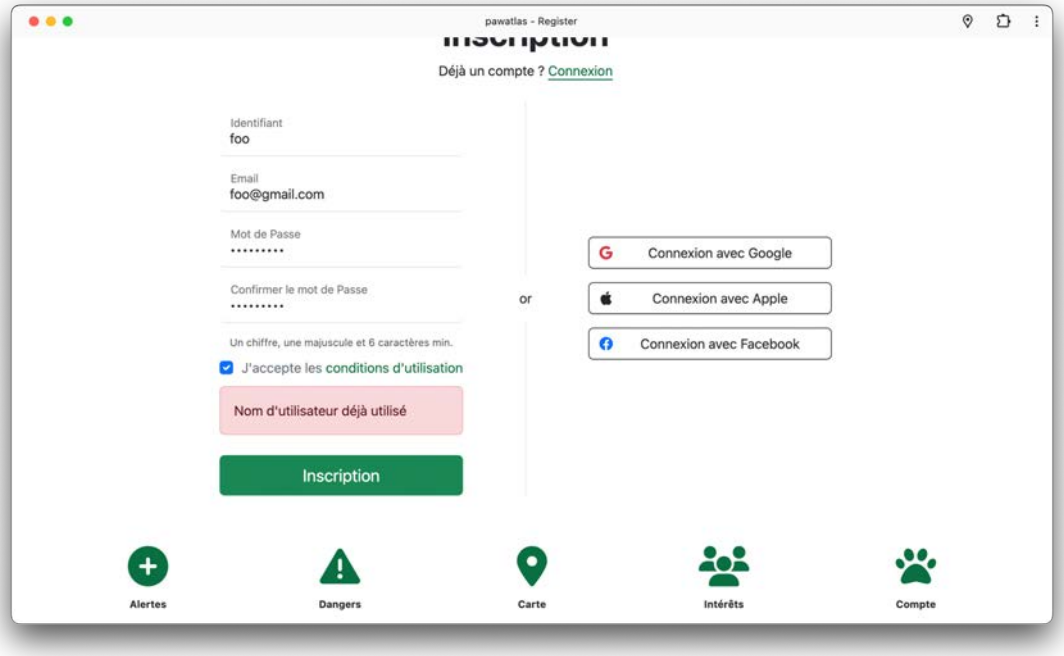

Figure 26 : Inscription - identifiant déjà utilisé

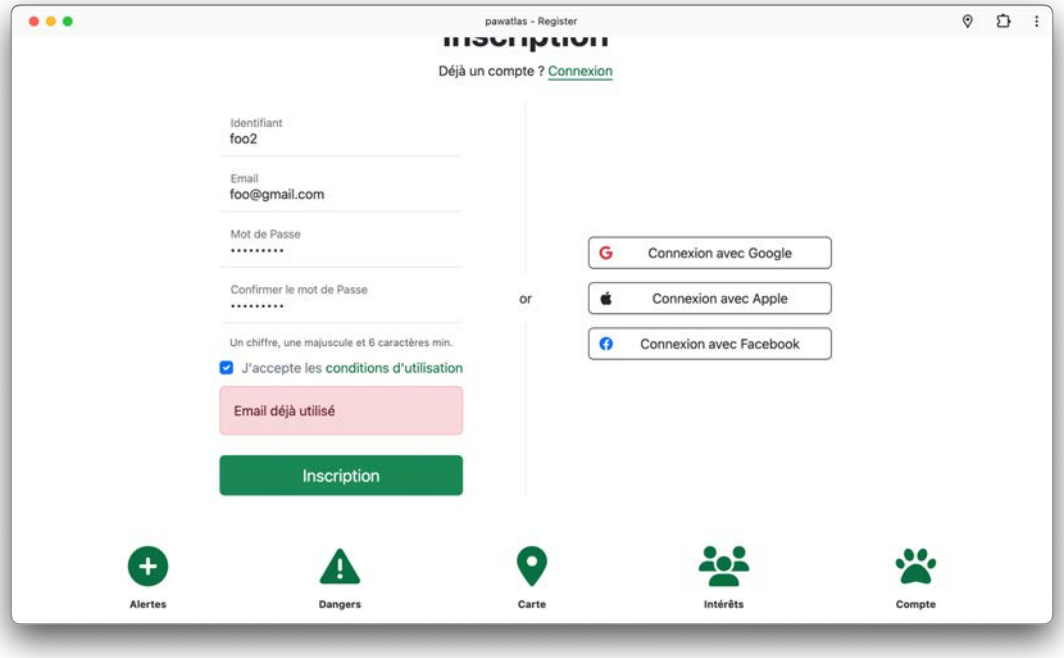

Figure 27 : Inscription - courriel déjà utilisé

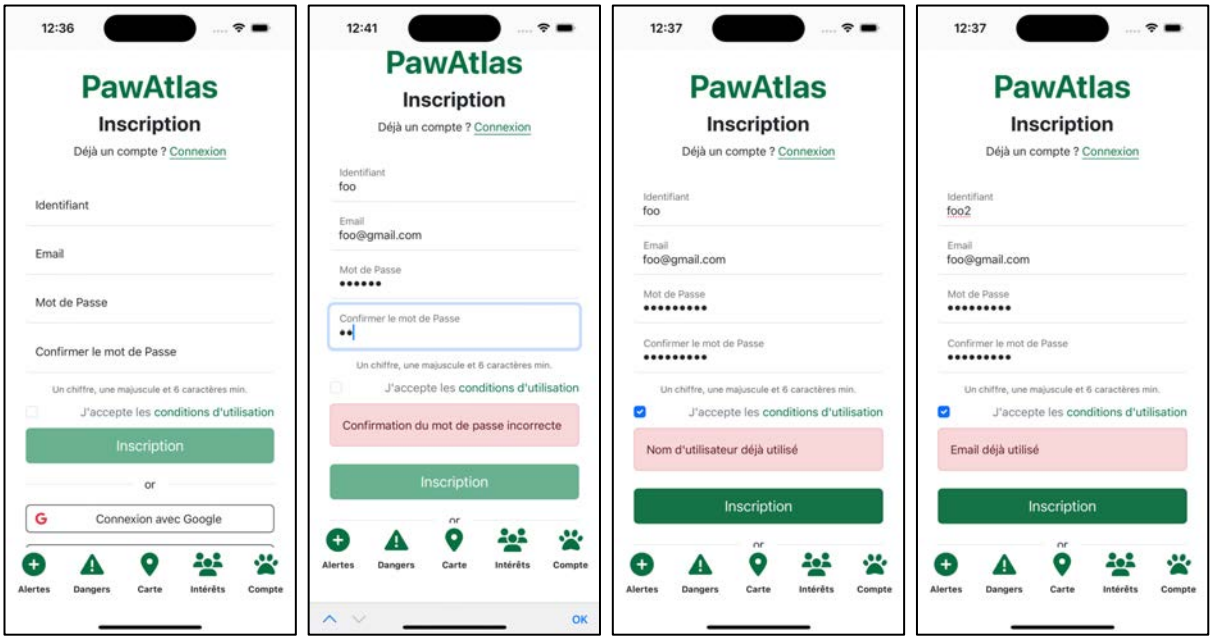

Figure 28 : Inscription - version mobile

3.6 Carte

La carte interactive est le cœur de l'application. Celle-ci fournit plusieurs fonctionnalités pour l'utilisateur. Elle provient d'OpenStreetMap [17] et a été intégrée à l'application grâce à la librairie en usage libre Leaflet [18].

Notons aussi que pour utiliser les fonctionnalités de localisation, l'utilisateur doit les accepter lorsque l'application lui le demande. Ceci ne se fait en principe qu'une seule fois par appareil utilisé. Les images présentées ci-après impliquent que cet accord a été donné auparavant.

Afin d'alimenter le texte, les images insérées montrent l'application telle qu'elle se présente sur un ordinateur. À la fin de cette section, un concentré permettra de visualiser celle-ci sur un mobile de type iOS.

3.6.1 Chargement

L'application se veut être fluide par son installation sur le périphérique de l'utilisateur. Cependant, lorsque l'utilisateur est redirigé sur la carte, il peut y avoir un temps relativement court où celle-ci affiche l'intégralité de la planète avant de se recentrer sur sa position. Ceci est lié à la géolocalisation qui peut prendre quelques secondes.

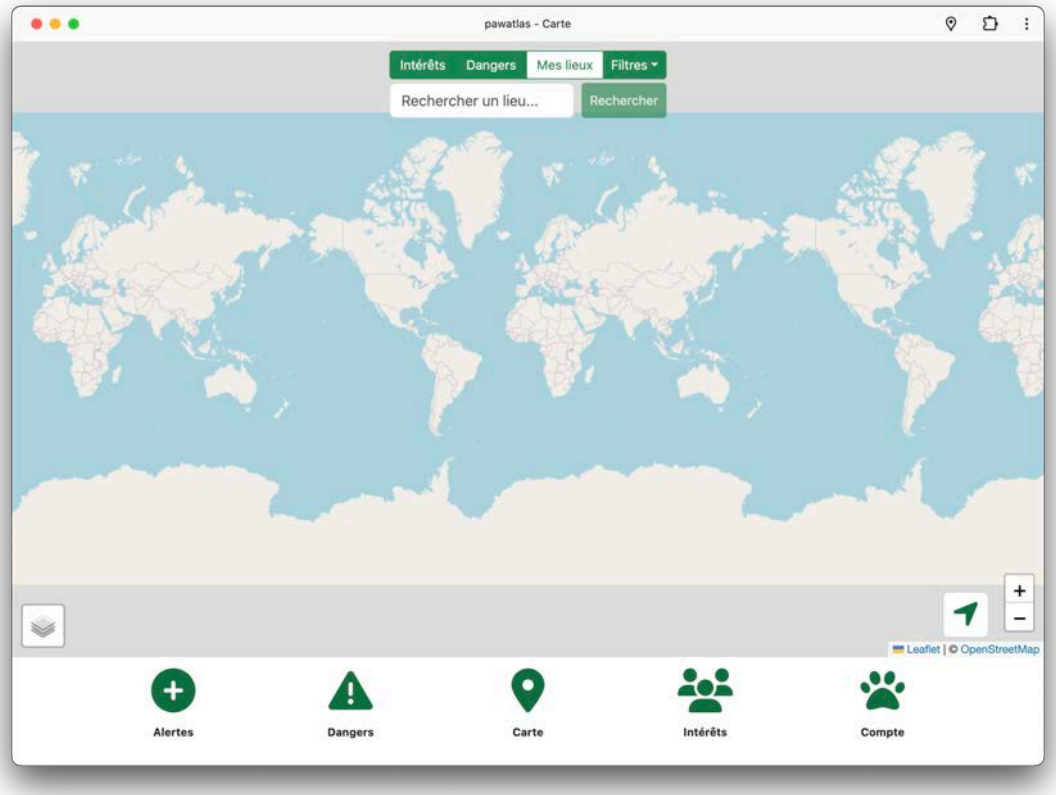

Figure 29 : Chargement de la carte - la localisation est en cours

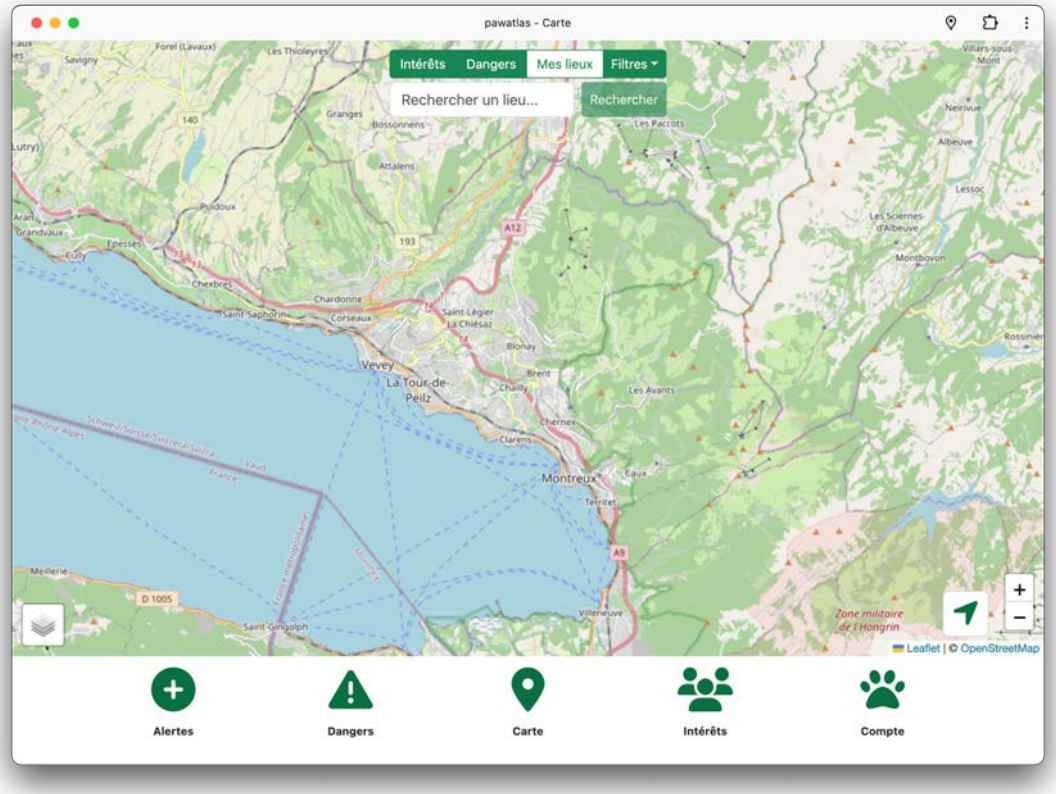

Figure 30 : Chargement de la carte - la localisation est terminée

3.6.2 Contrôle de la carte

Une fois la carte affichée, l'utilisateur peut zoomer ou agrandir celle-ci grâce à la molette de la souris, avec ses doigts sur mobile, ou encore grâce aux boutons « + » et « - » situés en bas à droite de l'écran.

L'utilisateur peut aussi appuyer sur l'icône se trouvant à proximité du zoom pour se localiser. Cela va créer un cercle bleu sur la carte qui va s'ajuster sur sa position.

En bas à gauche, l'utilisateur peut changer le design de la carte en choisissant ce qu'on appelle un « layer ». Ceux-ci sont en usage libre [19] et seront certainement modifiés par la suite selon les préférences de l'entreprise E-ProShop Sàrl.

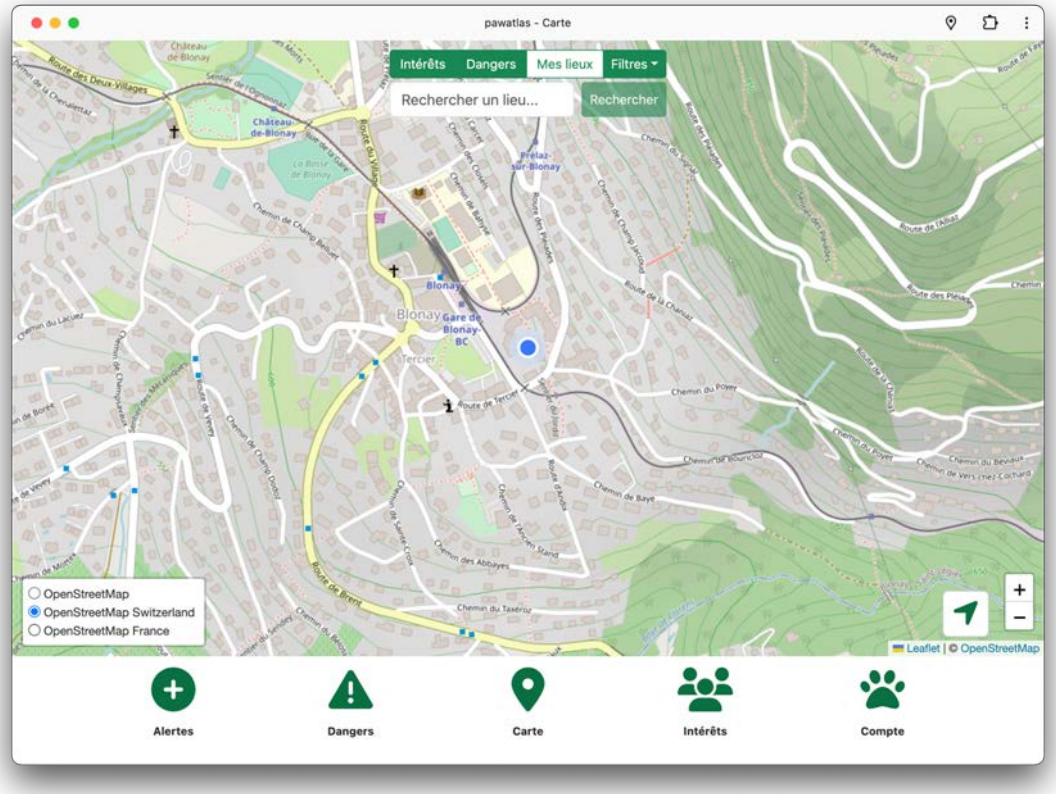

Figure 31 : Carte centrée sur la position de l'utilisateur avec le layer OSM Switzerland

3.6.3 Ajout d'un lieu

En cliquant sur la carte, l'utilisateur verra un popup apparaitre lui proposant d'ajouter un lieu. S'il continue, un formulaire s'affichera au centre de l'écran. L'utilisateur peut voir les coordonnées géographiques de l'endroit où il a cliqué. Il peut choisir un type de lieu entre « Danger » ou « Intérêt » et sélectionner une catégorie. Il est ensuite prié d'inscrire une petite description pour valider le formulaire.

L'ajout d'une image est facultatif. Cliquer sur « Choisir un fichier » ouvrira les dossiers de l'ordinateur et seuls les fichiers comprenant une image seront sélectionnables. Sur mobile, l'utilisateur peut même directement prendre une photographie.

Un fois le formulaire validé, il se ferme et le lieu apparait instantanément sur la carte avec un icône correspondant à la catégorie sélectionnée.

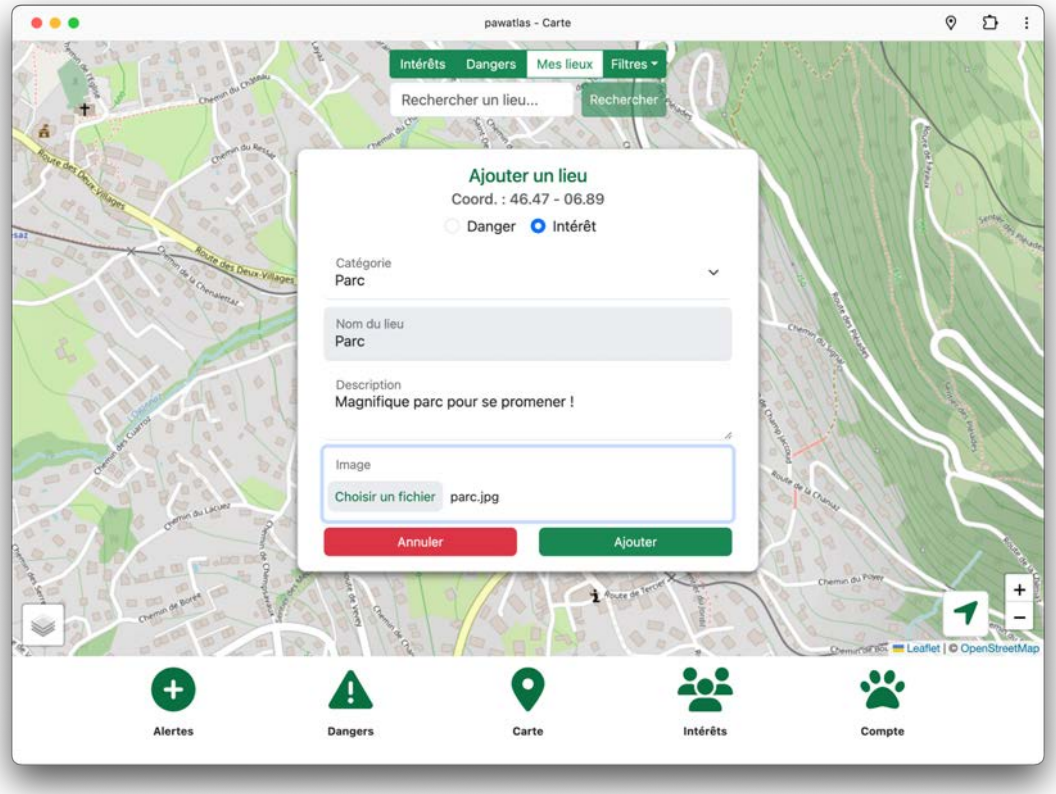

Figure 32 : Ajout d'un lieu sur la carte

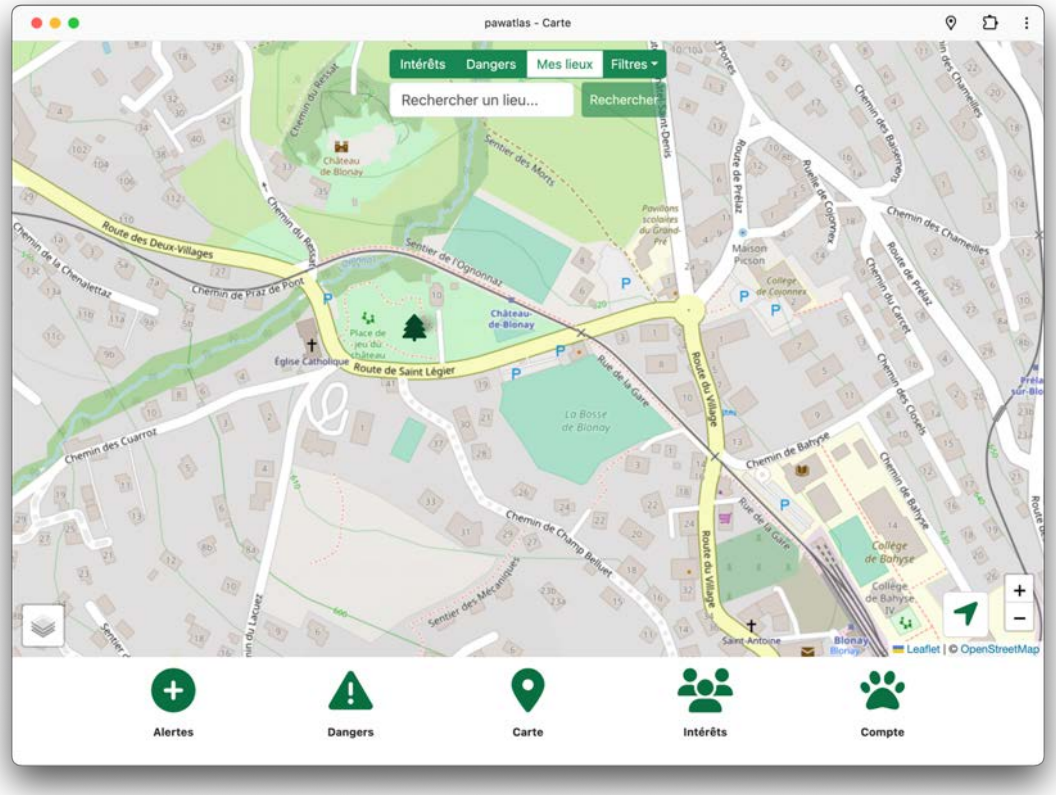

Figure 33 : Un lieu a été ajouté à la carte

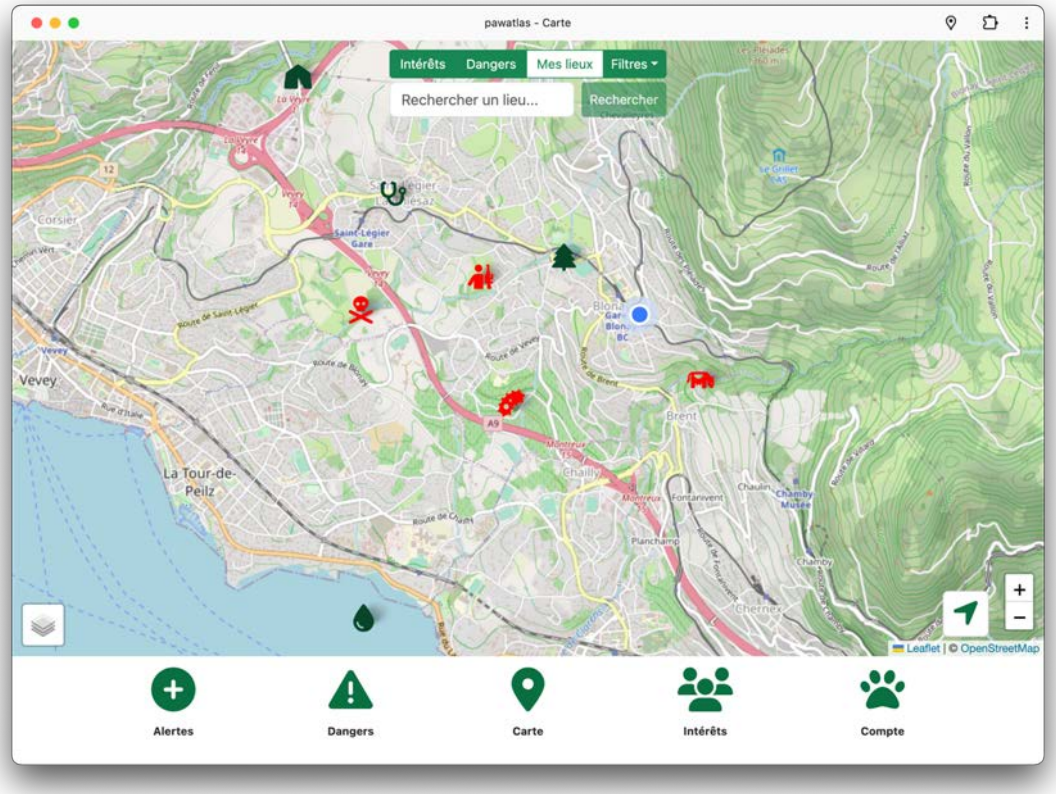

Figure 34 : Plusieurs lieux ont été ajoutés à la carte

3.6.4 Affichage des lieux

Au sommet de la page, un menu permet à l'utilisateur de choisir quels lieux sont affichés. Il peut décider de voir uniquement les points d'intérêts, uniquement les dangers, ou alors les deux types. Il peut aussi faire apparaitre seulement les lieux qu'il a créé lui-même et un système de filtres permet de sélectionner les catégories.

Une barre de recherche est présente. Nous reviendrons sur cette fonctionnalité un peu plus loin dans ce chapitre dans la section intitulée « Nominatim ».

Concernant l'affichage des points, si nous agrandissons la zone affichée sur carte, les icônes auront tendance à se chevaucher. C'est pourquoi un système de tri a été mis en place. Plus un lieu a été apprécié, plus il sera visible avec un niveau de zoom faible. Pour les points qui ne comprennent que peu de votes, ils seront visibles uniquement lorsque la carte est centrée sur une zone réduite.

Cependant, il est très difficile de décider à partir de combien de votes un lieu doit être affiché selon un certain degré de zoom. Cela va dépendre du nombre d'utilisateurs actifs. C'est pourquoi nous avons implémenté cette fonctionnalité de manière arbitraire et l'ajusterons par la suite.

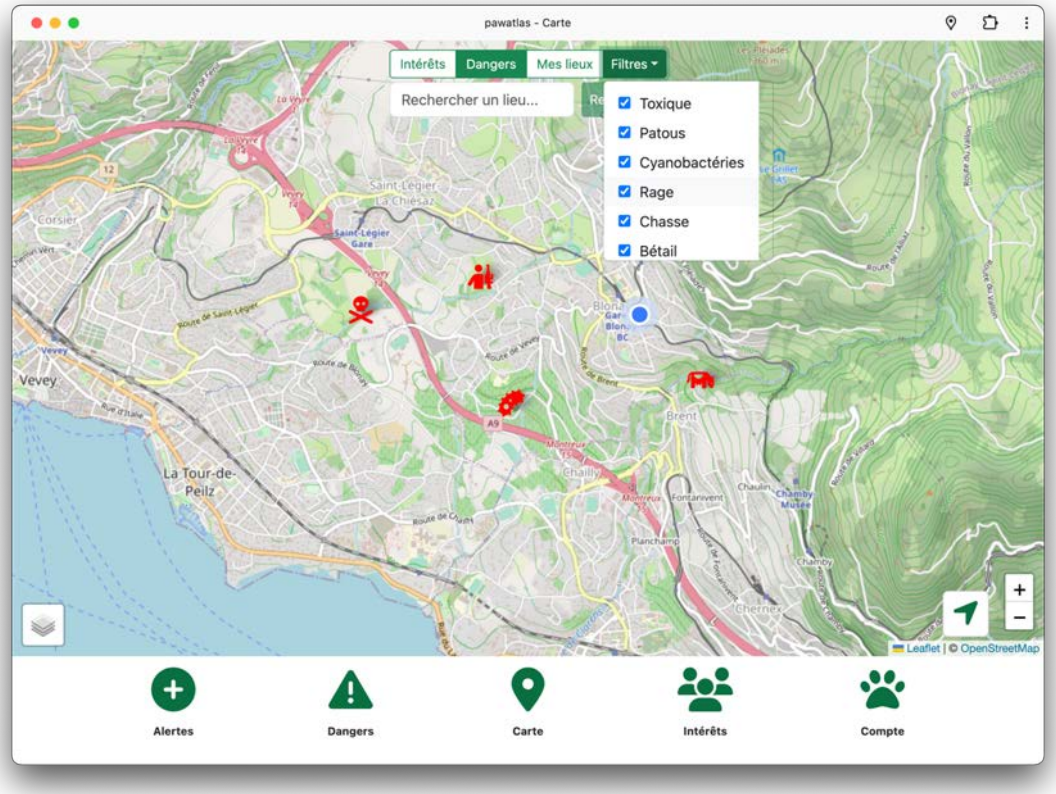

Figure 35 : Carte avec uniquement les dangers affichés

3.6.5 Détails d'un lieu

Cliquer sur une icône permet d'afficher un popup indiquant le titre du lieu ainsi que son nombre de votes. Un bouton permet d'afficher ses détails, ce qui fait apparaître son image si elle existe, ainsi que différentes informations comme sa date de création et sa distance à vol d'oiseaux par rapport à la position de l'utilisateur.

Un lien « informations » permet d'en savoir plus sur ce lieu. Nous analyserons cette fonctionnalité dans la section suivante en même temps que celle liée à la barre de recherche.

Deux icônes permettent d'apprécier ou de rejeter le point. Si un lieu obtient un certain nombre de voix négatives (actuellement une seule pour cette démonstration), il est mentionné à l'utilisateur que son vote engendrera la suppression de celui-ci et un message de confirmation lui sera adressé à l'étape suivante. S'il continue, le point est simplement supprimé de la carte et de la base de données.

Des boutons permettent de supprimer et de modifier le lieu. Dans le cas où un utilisateur tente d'effectuer ces actions sur un lieu qu'il n'a pas créé, une alerte apparaitra l'empêchant de continuer.

Modifier un lieu ouvrira le même formulaire utilisé lors de sa création. La seule différence réside dans le fait que nous pouvons choisir de supprimer son image.

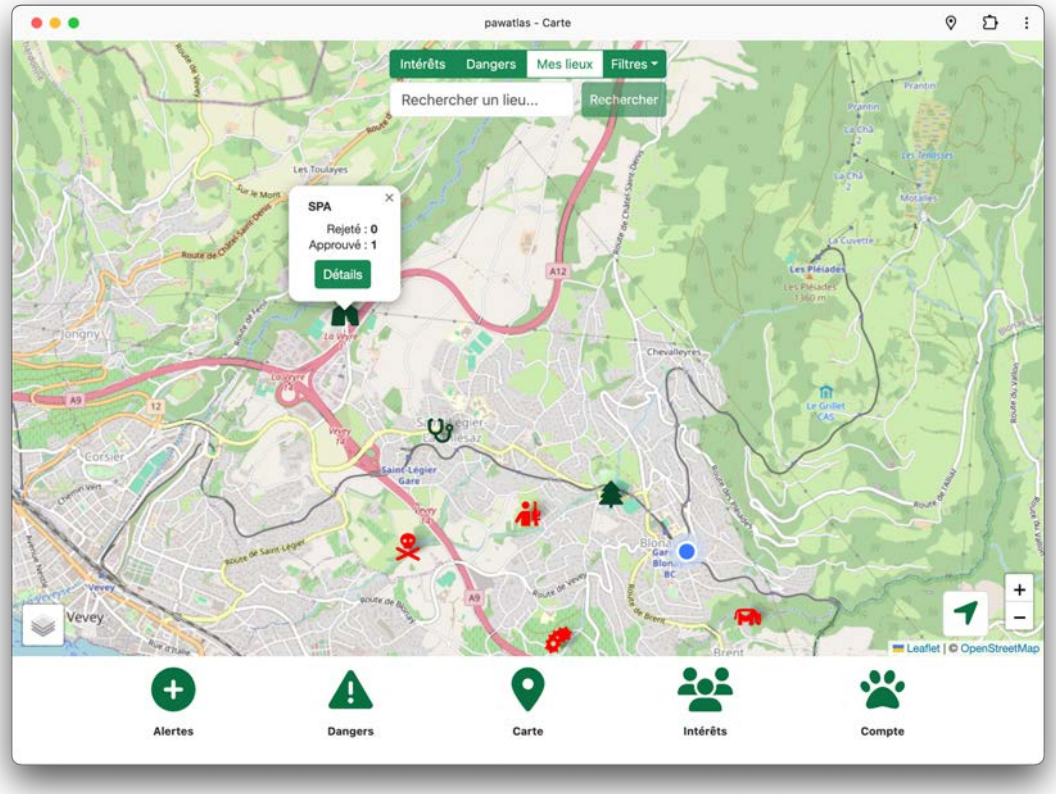

Figure 36 : Popup apparaissant au clic sur une icône

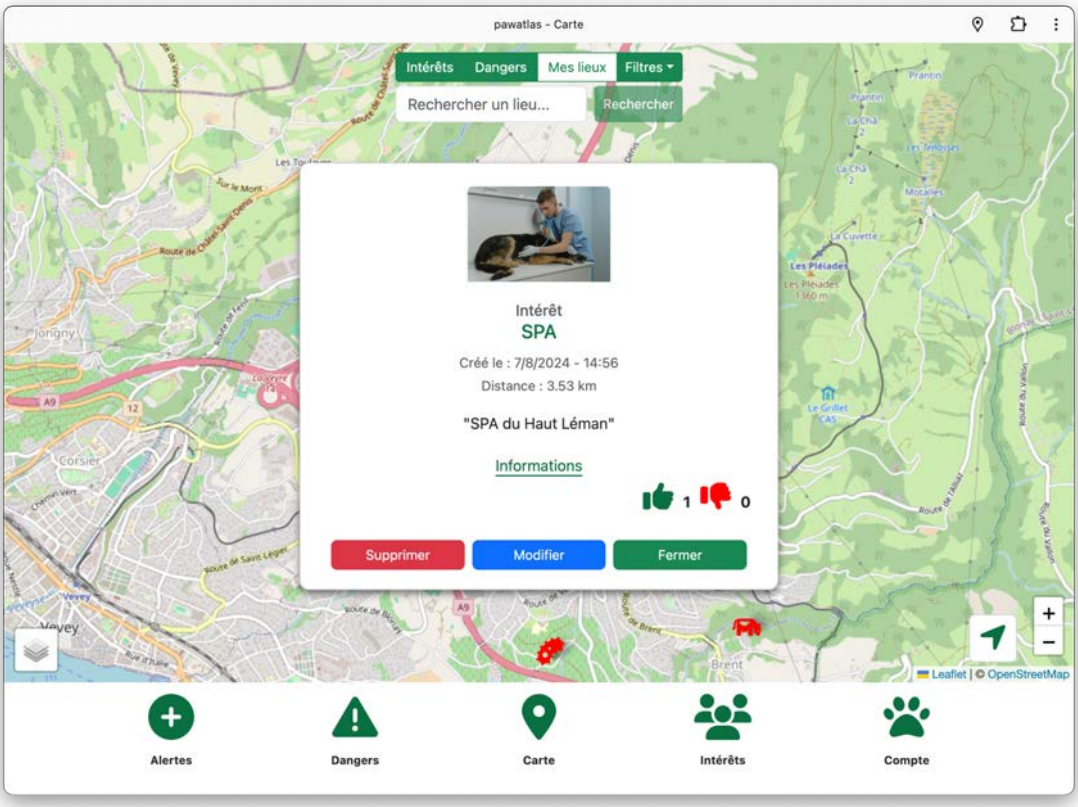

Figure 37 : Détails d'un lieu

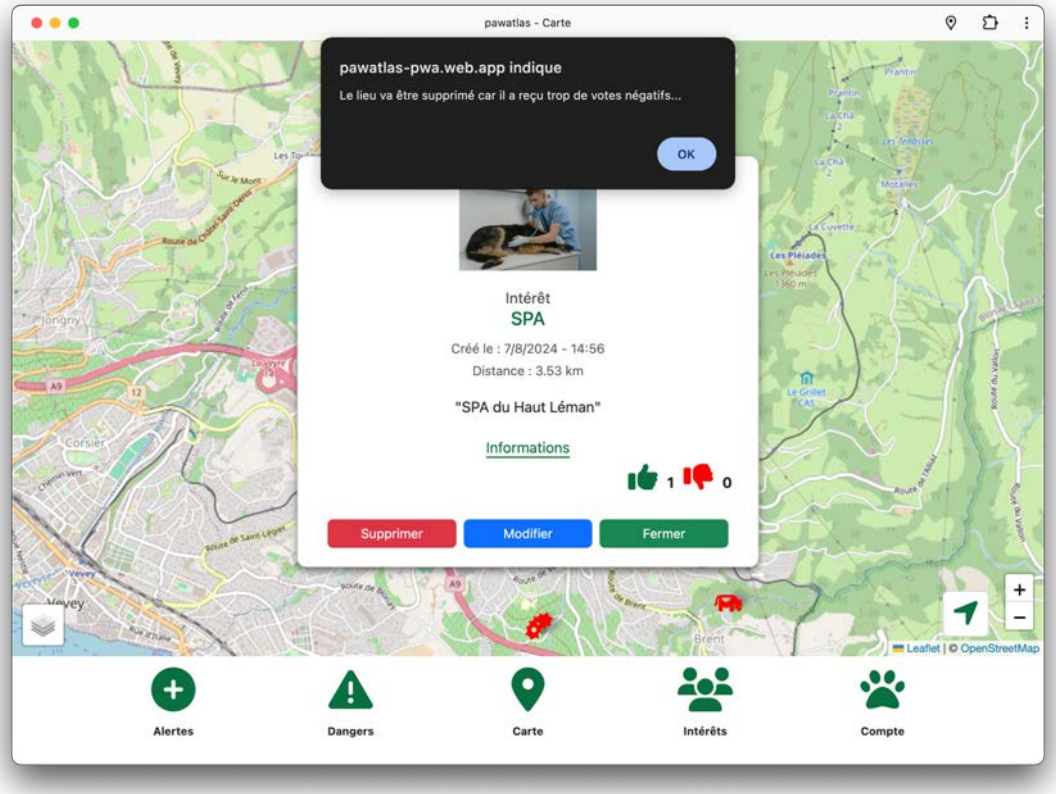

Figure 38 : Alerte informant qu'un certain nombre de votes négatifs supprime le lieu

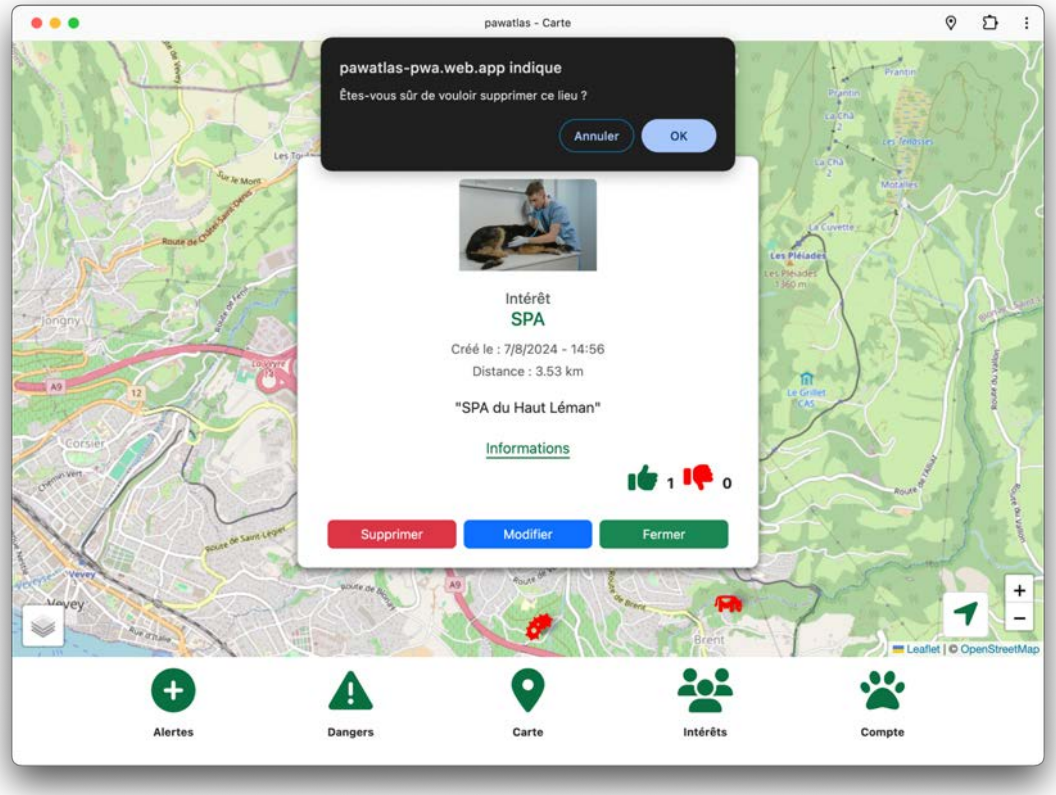

Figure 39 : Alerte confirmant la suppression d'un lieu

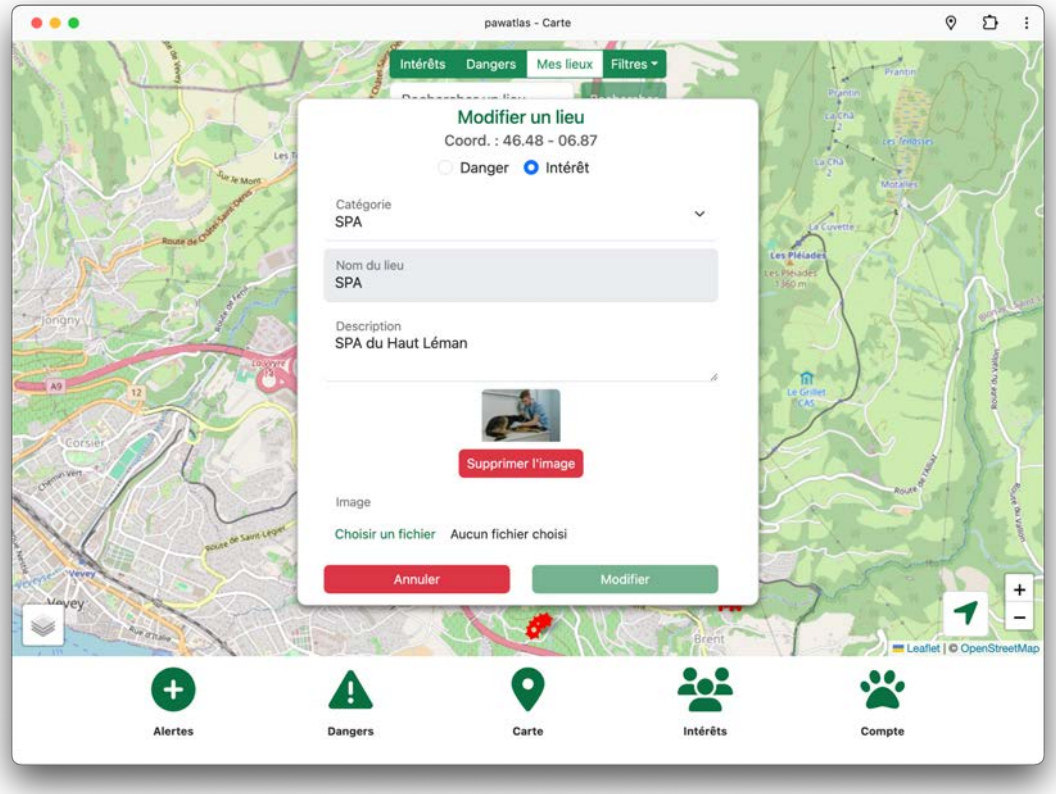

Figure 40 : Modification d'un lieu

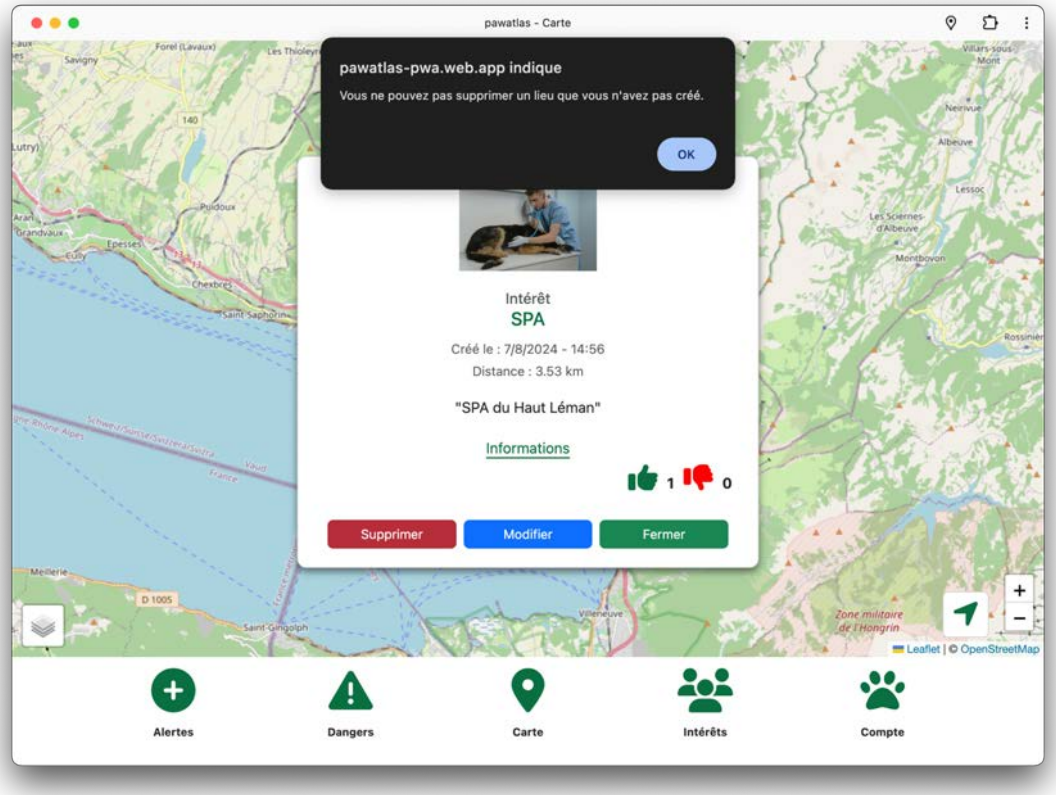

Figure 41 : L'utilisateur ne peut pas supprimer un lieu qu'il n'a pas créé

3.6.6 Nominatim

Comme mentionné, une barre de recherche est présente dans le menu situé en haut de la carte et un lien « informations » est disponible lorsque nous affichons les détails d'un lieu.

Ces fonctionnalités sont possibles grâce à l'API Nominatim [20]. Celle-ci est un service gratuit qui utilise les données d'OpenStreetMap pour retrouver des coordonnées GPS par un nom ou par une adresse, et vice-versa.

Lorsque l'utilisateur affiche les détails d'un lieu sur la carte de notre application, il peut cliquer sur le lien « informations ». Il sera ensuite redirigé sur une page que nous avons créée et qui affiche les données récupérées de l'API Nominatim. Il y verra le type de localisation, sa catégorie, son nom, son adresse et ses coordonnées géographiques. De plus, les crédits pour ce service sont affichés selon les règles d'utilisation de l'API [21]. L'utilisateur peut ensuite cliquer sur retour et cela le redirige sur la carte qui va se centrer sur le lieu affiché précédemment.

La recherche offerte au haut de la carte fonctionne aussi avec l'API Nominatim. L'utilisateur peut l'utiliser pour trouver un lieu et une liste de résultat s'affiche au centre de l'écran. Il peut ensuite cliquer sur le résultat qui l'intéresse et cela va centrer la carte sur ce point.

Notons toutefois que ce service n'est pas très précis en comparaison à celui de Google Map Places [22], qui est malheureusement payant. Toutefois, il a permis de faire des essais et d'implémenter la structure de base dans notre code pour ce genre de fonctionnalités.

Afin de se faire une idée de ce service et de découvrir les résultats avancés d'une recherche, nous invitons le lecteur à se rendre sur la page officielle de l'API Nominatim où il est possible de l'utiliser : https://nominatim.openstreetmap.org/ui/search.html

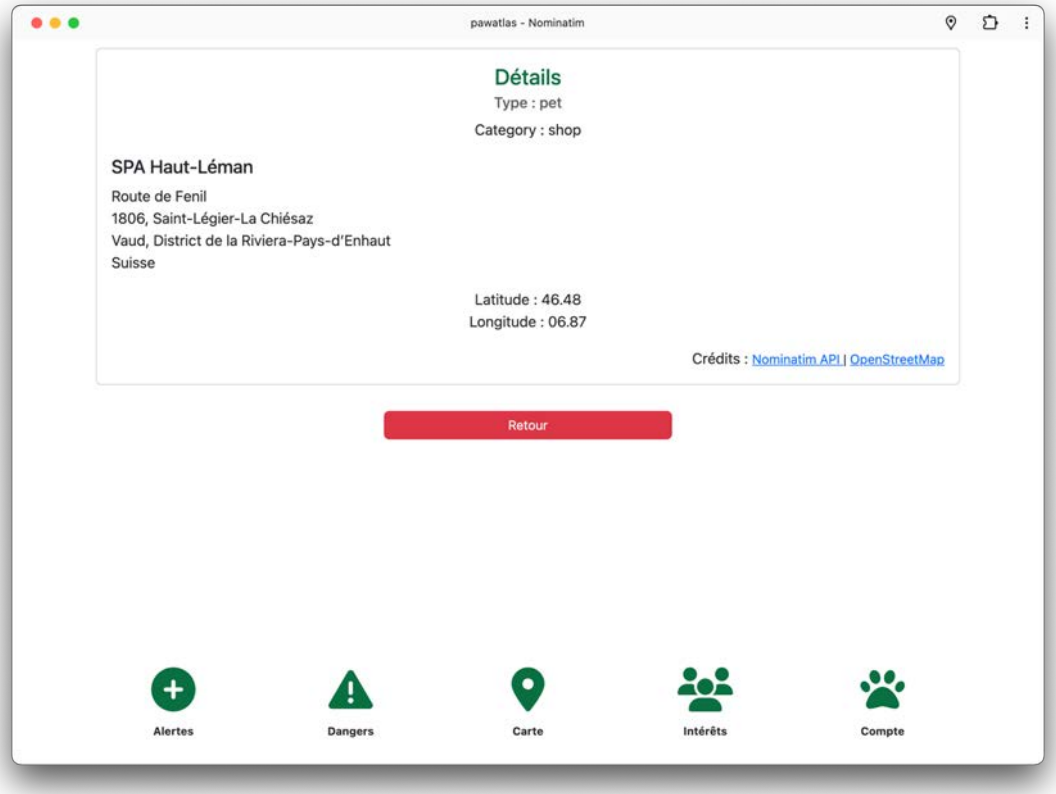

Figure 42 : Informations sur un lieu avec l'API Nominatim

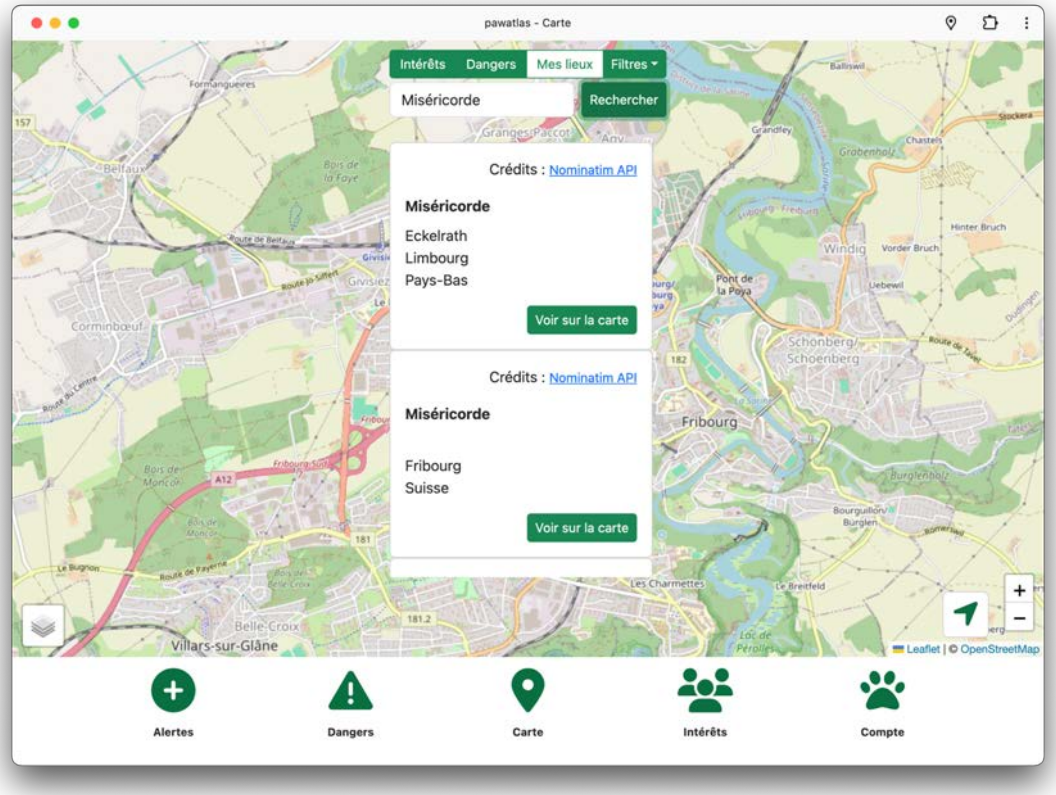

Figure 43 : Recherche de lieu avec l'API Nominatim

3.6.7 Utilisation de la carte sur iOS

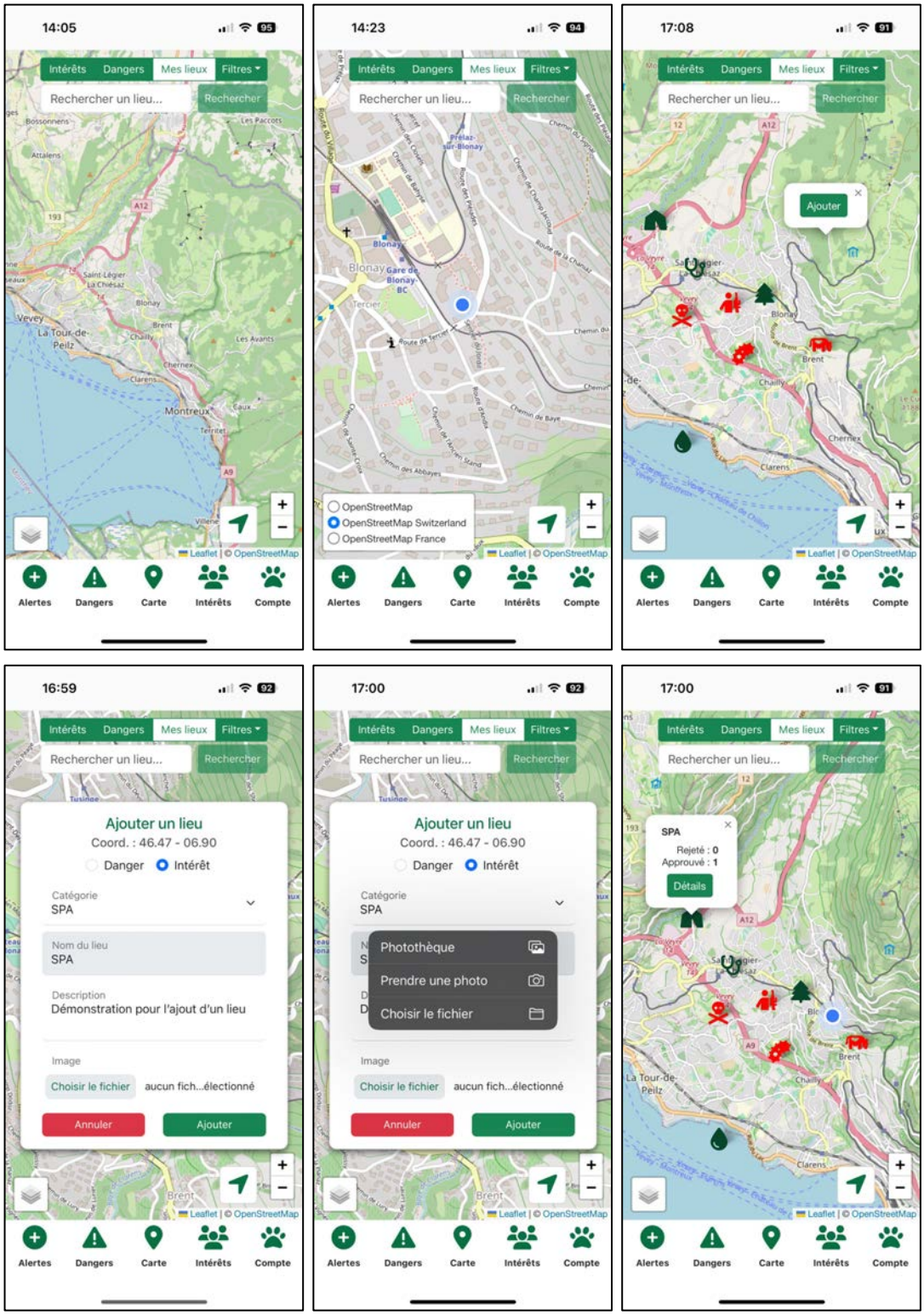

Figure 44 : Fonctionnalités de la carte - version mobile

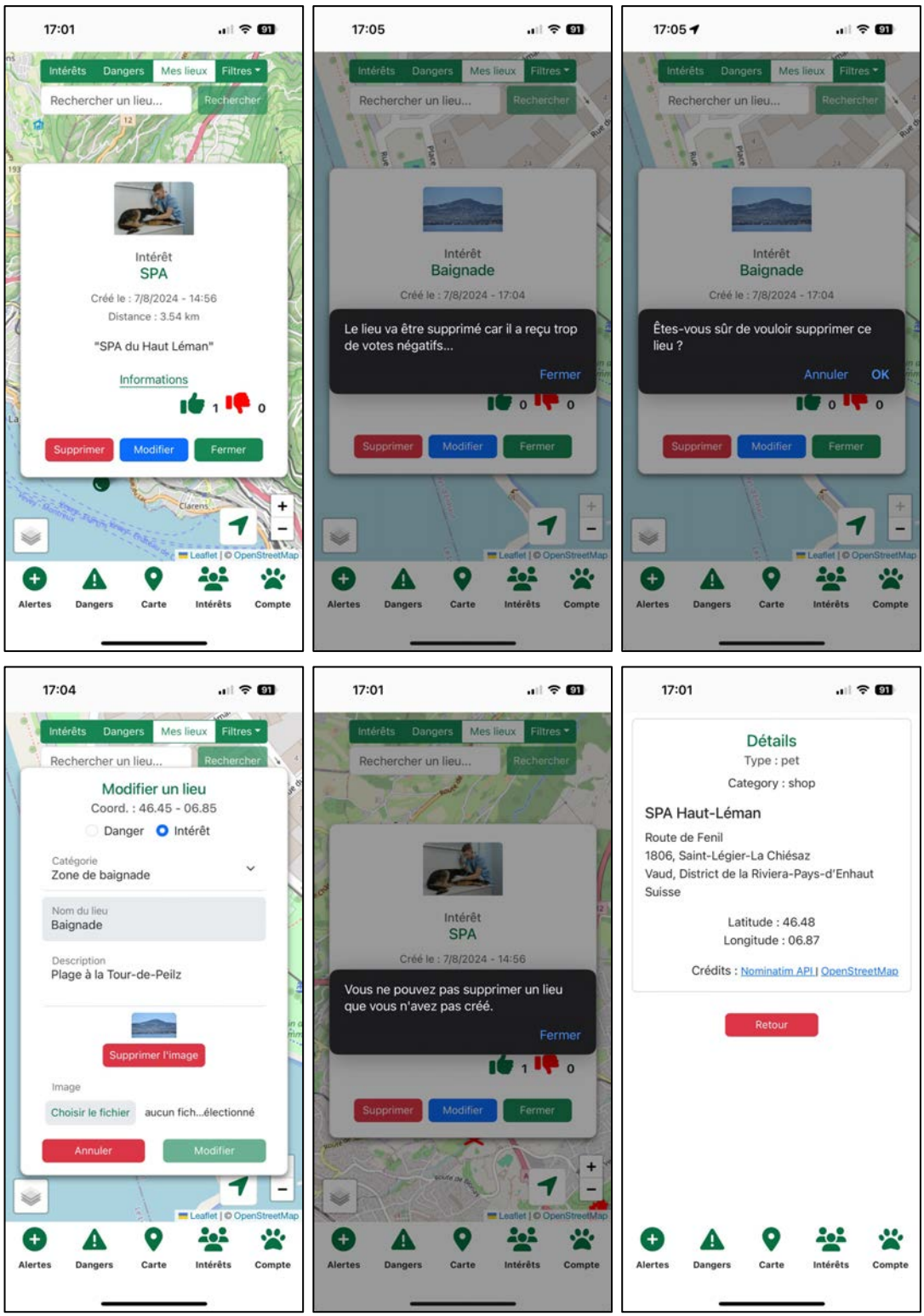

Figure 45 : Fonctionnalités de la carte - version mobile - 2

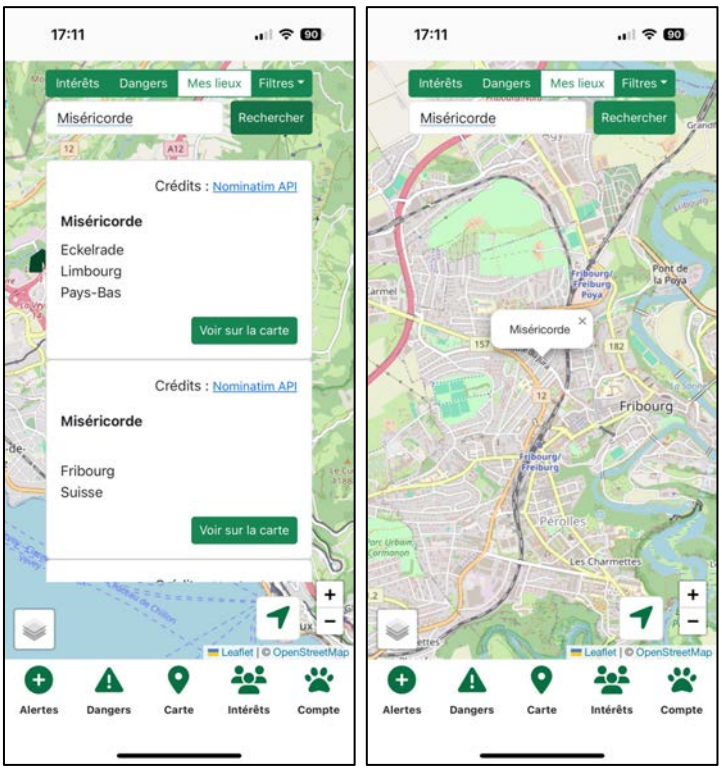

Figure 46 : Fonctionnalités de la carte - version mobile - 3

3.7 Dangers

Un cas d'utilisation consistait à consulter la liste des alertes ajoutées à la carte. Ces alertes ont été implémentés comme étant les lieux qui comprennent le type « Danger ».

Pour y accéder, l'utilisateur peut cliquer sur l'icône correspondant dans le menu de navigation situé en bas de l'écran.

Un menu est présent en haut de cette page pour filtrer et trier les dangers. Le tri peut se faire par date, par distance par rapport à la position actuelle de l'utilisateur, par votes positifs ou encore par votes négatifs.

Il est possible de sélectionner un danger pour le supprimer ou le modifier. L'utilisateur peut aussi décider de l'afficher sur la carte.

Relevons que nous n'avons pas implémenté la possibilité de sélectionner le rayon des alertes présentes dans la liste. En effet, nous avons pensé que la fonctionnalité de tri par distance était suffisante pour le prototype.

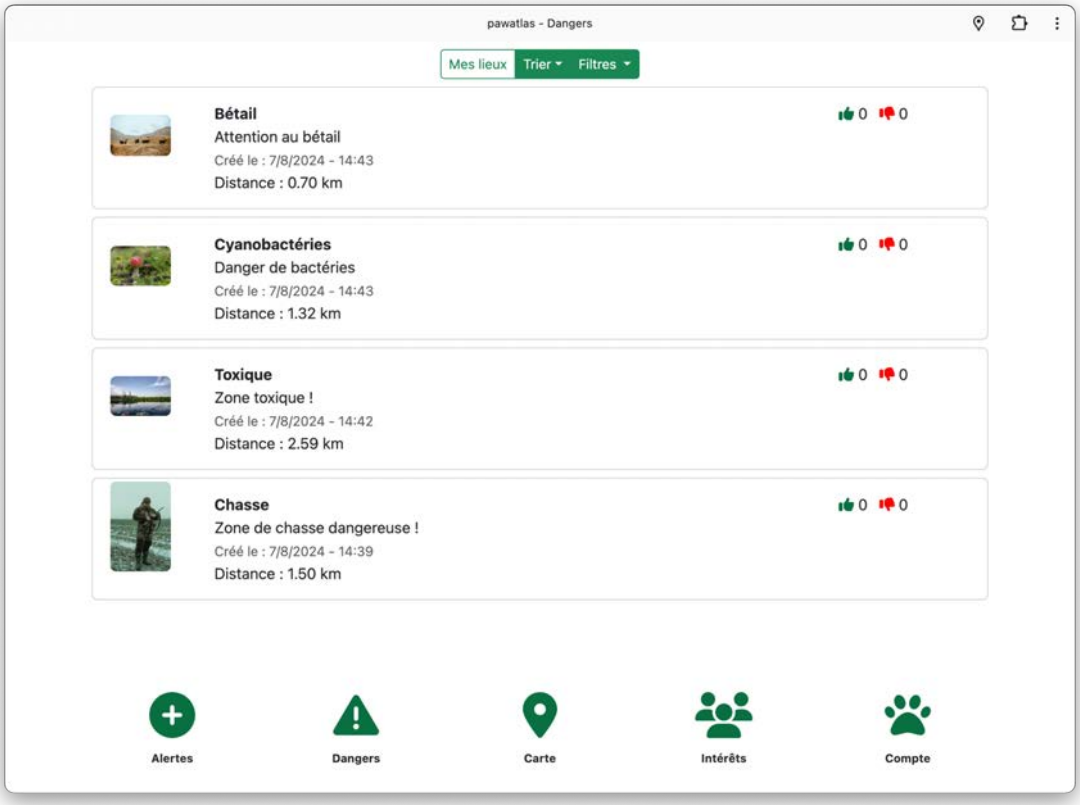

Figure 47 : Affichage des dangers

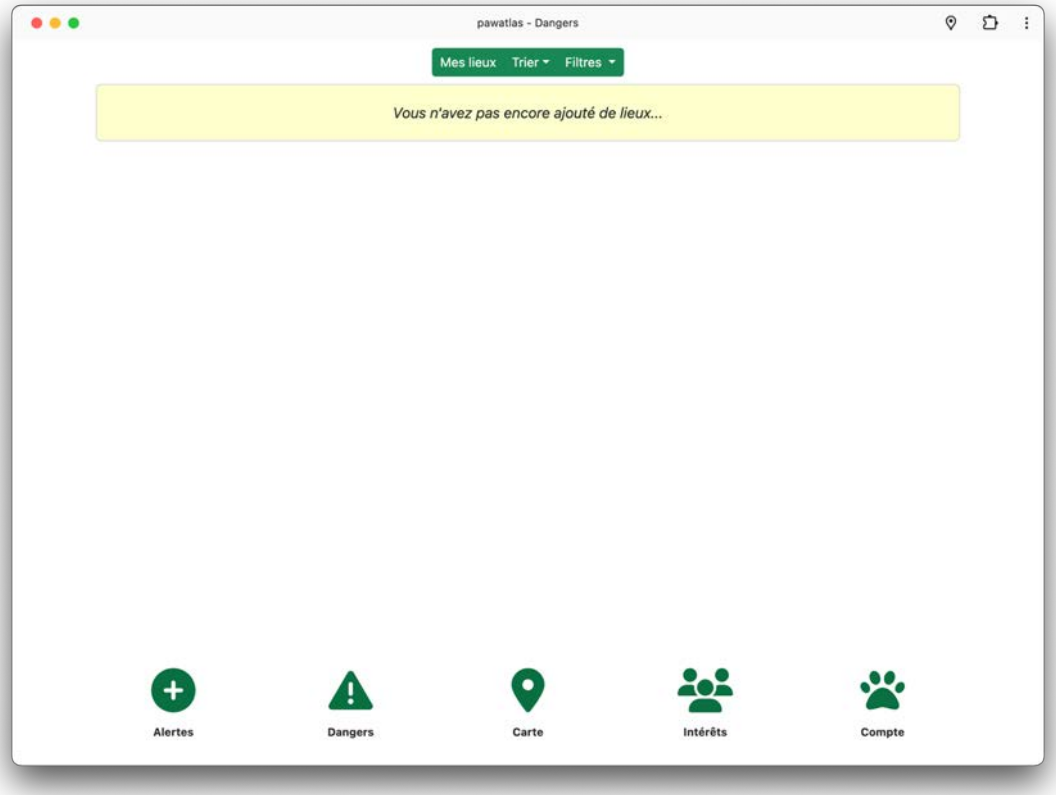

Figure 48 : Dangers - l'utilisateur n'a pas encore ajouté de lieux

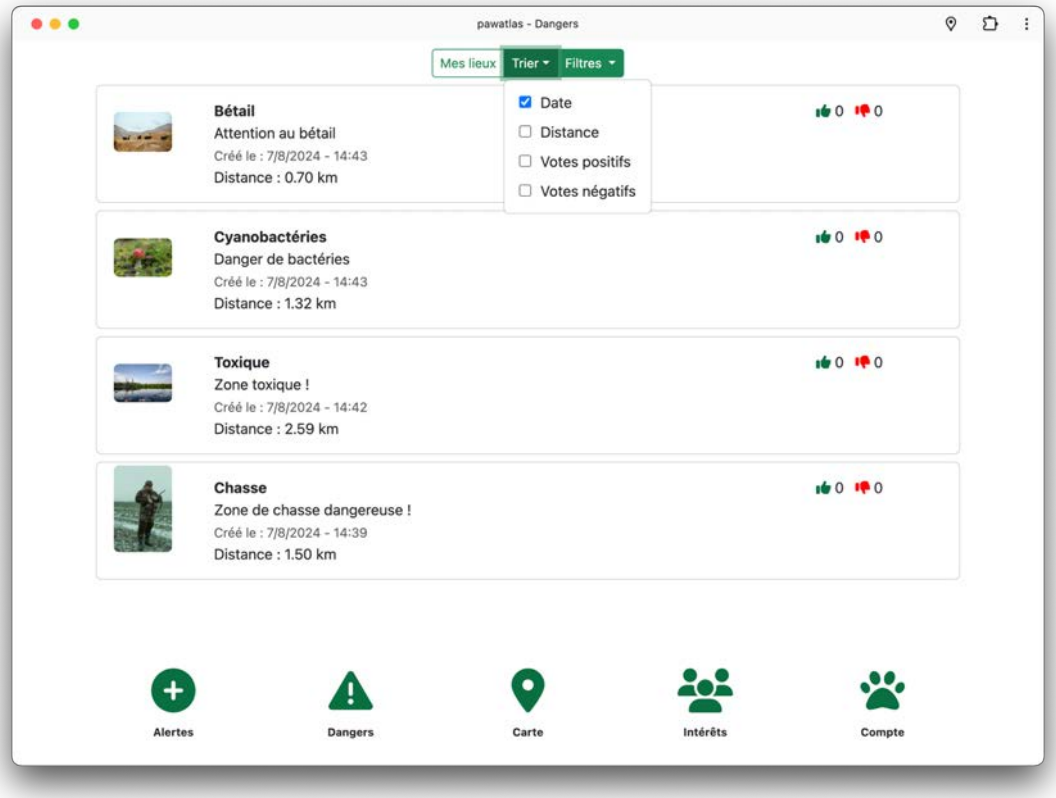

Figure 49 : Dangers - tri

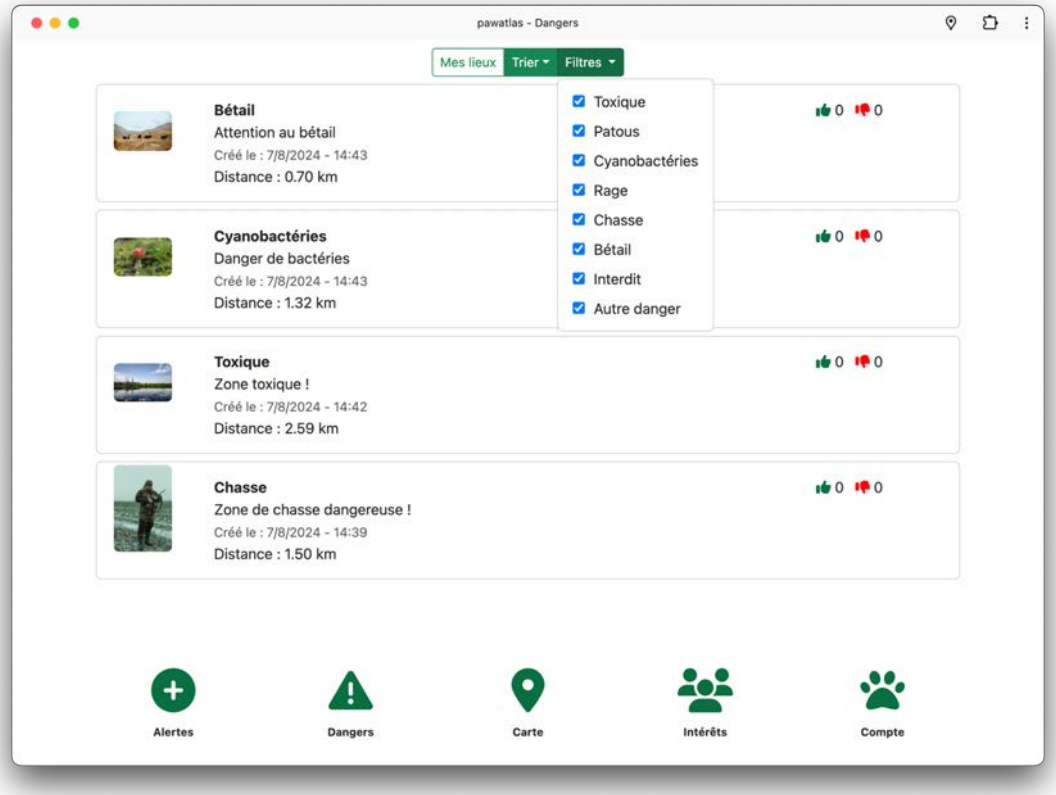

Figure 50 : Dangers - filtres

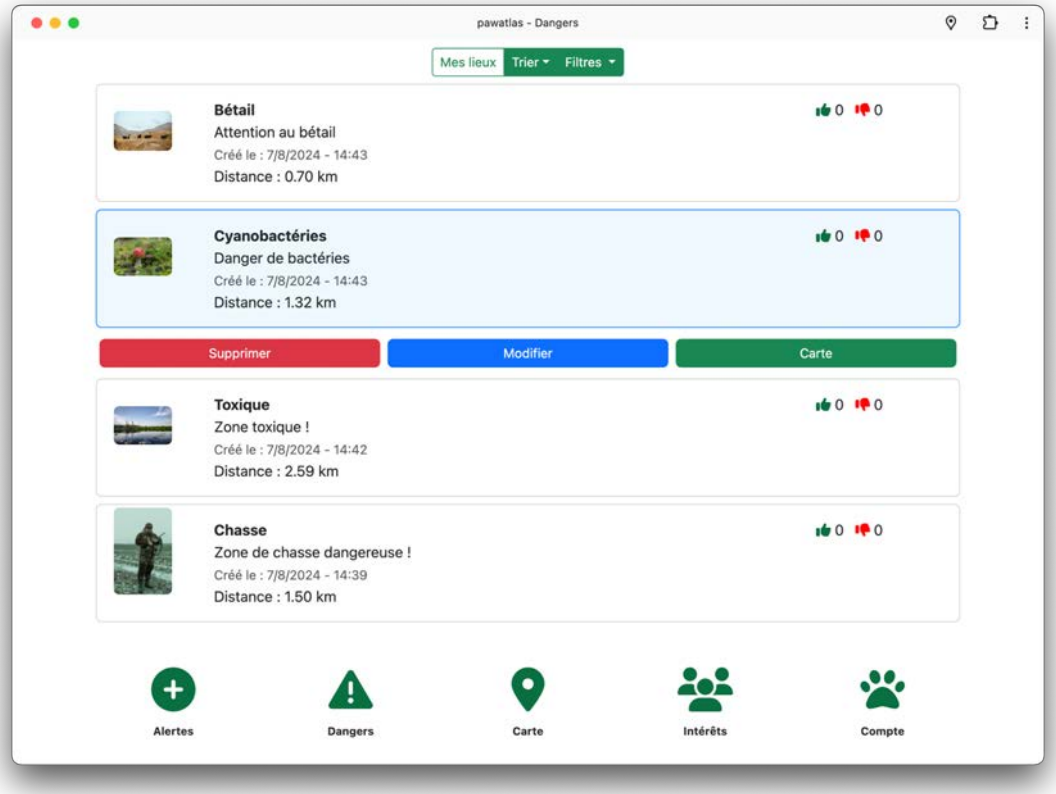

Figure 51 : Dangers - sélection

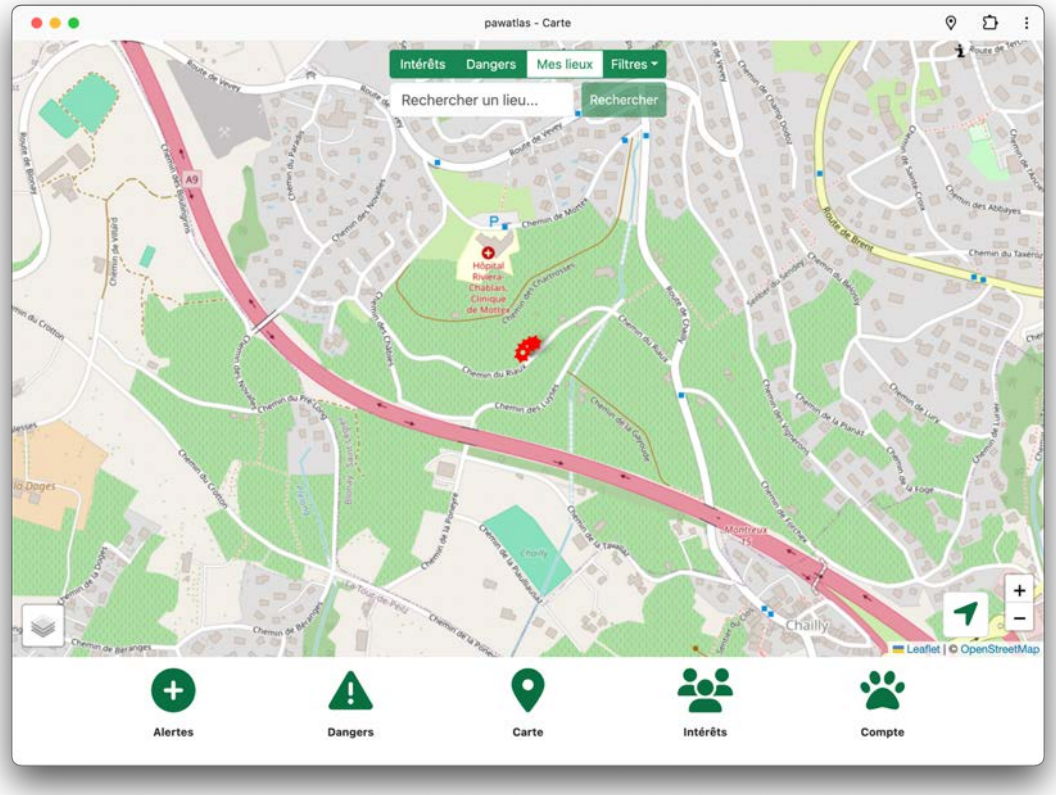

Figure 52 : Dangers - redirection sur la carte

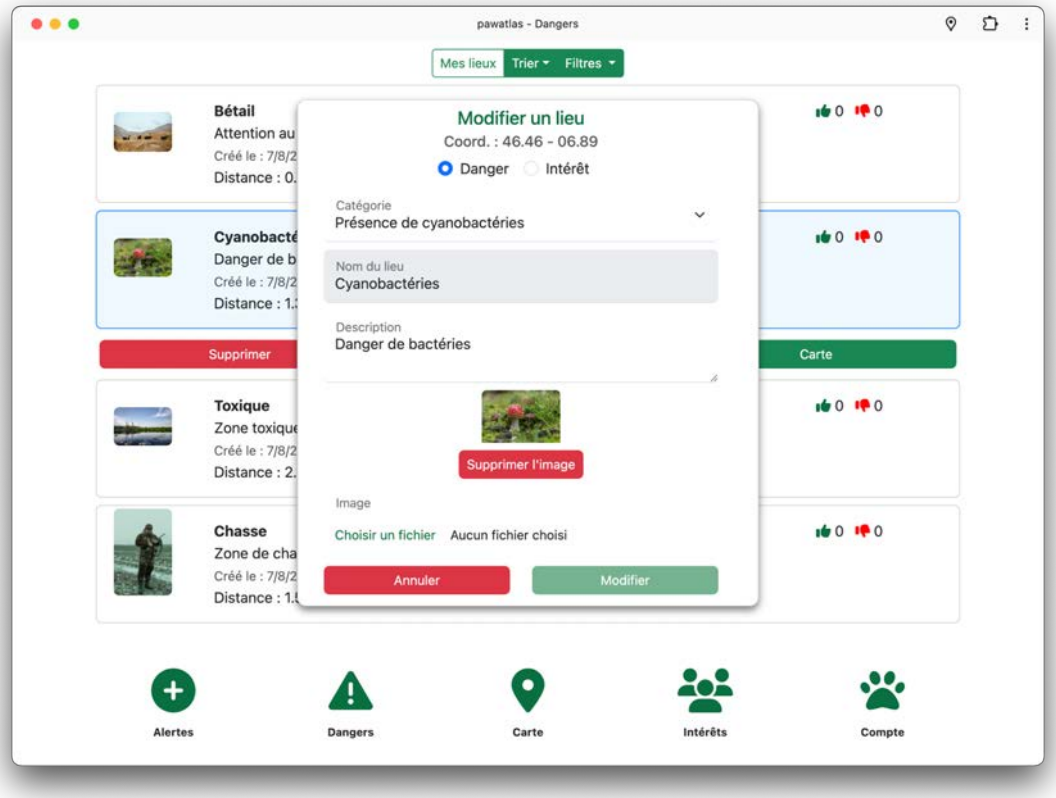

Figure 53 : Dangers - modification d'un lieu

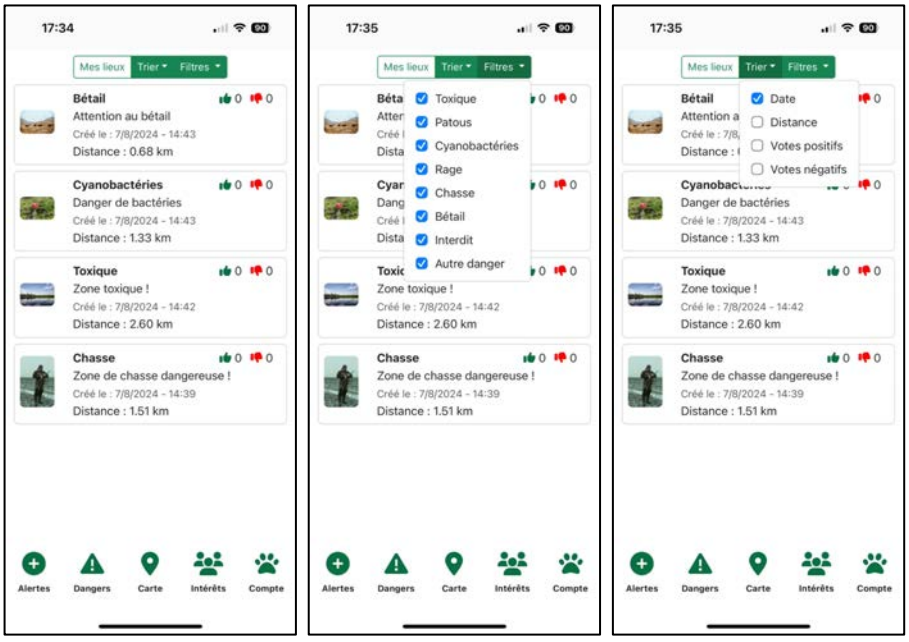

Figure 54 : Dangers - version mobile

3.8 Intérêts

La page consacrée aux lieux indiquant une zone d'intérêt est accessible depuis le menu de navigation. Cette page ne fait pas partie des cas d'utilisation nécessitant une implémentation pour le prototype, elle est donc provisoire.

Toutefois, nous avons décidé de réutiliser le composant Angular permettant à l'utilisateur d'afficher la liste des dangers. Ainsi, les mêmes fonctionnalités sont disponibles, cette fois pour les intérêts.

3.9 Alertes et Compte

Ces deux pages ne sont pas implémentées pour le prototype consacré à ce travail de Bachelor. Pour la page « Alertes », l'idée de l'entreprise E-ProShop Sàrl serait de développer les fonctionnalités de notifications en temps réel afin qu'un utilisateur puisse avertir les utilisateurs qui se situent à proximité si son animal de compagnie est perdu. La page « Compte » permettrait à l'utilisateur de gérer son compte, de créer un profil personnalisé et de gérer celui de ses animaux. Concernant le fil d'actualité communautaire, cette idée est en suspens.

4

Point de vue développeur

4.1 Architecture

L'objectif de ce chapitre est de parcourir l'ensemble du code réalisé pour le prototype. Nous allons dans un premier temps analyser ce qu'on appelle le Frontend, soit toute la partie de l'application qui est exposée au client. Ensuite, nous verrons l'infrastructure Backend qui concerne la partie côté serveur. Nous terminerons par la base de données.

L'architecture choisie pour ce projet est fortement liée au fait qu'aucun budget n'était disponible pour la réalisation de l'application. Il n'était donc pas possible d'héberger un serveur ou encore une base de données si cela engendrait des frais. C'est pourquoi nous nous sommes orientés vers les services de Firebase qui sont gratuits jusqu'à un certain stade d'utilisation [23].

De cette manière, l'architecture de notre projet est de type « serverless ». Elle diffère ainsi d'une architecture traditionnelle où un serveur est utilisé pour effectuer, par exemple, des requêtes auprès d'une base de données, ou encore pour fournir tout autre contenu à un client qui l'interroge.

Cependant, affirmer que notre architecture ne contient pas de serveur serait une erreur d'imprécision. En réalité, nous utilisons ceux de Firebase à travers des services. Voici un schéma qui résume bien cette idée [24] :

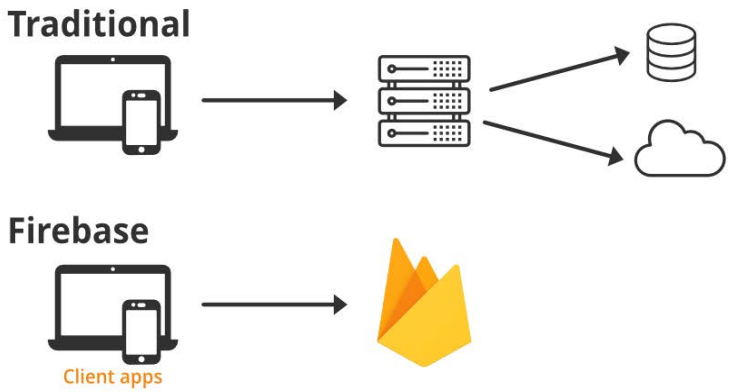

Figure 55 : Architecture du projet sans serveur

4.2 Frontend

Concernant la partie Frontend, nous avons fait le choix d'utiliser le framework Angular. Olivier Hondermarck définit ce terme de la façon suivante :

« En informatique, le terme framework pourrait être traduit, assez littéralement, par « socle d'application » ou « infrastructure de développement ». Il désigne un ensemble de pratiques, de composants et d'outils utilisés pour structurer et développer un projet web. Le but d'un framework est souvent de simplifier et d'accélérer le développement en apportant des briques fonctionnelles prêtes à l'emploi, comme la manipulation de la base de données, la sécurisation des saisies utilisateur, la création de templates… » [25]

4.2.1 Angular

Angular a été créé en 2010 par Google. Aristeidis Bampakos est un développeur reconnu officiellement par l'entreprise comme « Google Developer Expert ». Il le définit ainsi :

« Angular is a popular and modern JavaScript framework that can run on different platforms additional to the web, such as desktop and mobile. Angular applications are written in TypeScript, a superset of JavaScript that provides syntactic sugar such as strong typing and object-oriented techniques. » [26]

En d'autres termes, ce framework permet de développer des applications en utilisant le langage TypeScript, ce qui impose un typage des données et une approche orientée objet. C'est pour ces raisons que nous nous sommes tournés vers cet outil. Relevons que ce choix n'a pas rendu le développement de notre application plus facile. En effet, comme le mentionne Olivier Hondermarck à propos d'Angular :

« Son apprentissage implique de maîtriser TypeScript et une structure de fichiers et de modules vraiment lourde. » [25]

Cependant, c'est bien la structure et le langage imposés par le framework qui nous a convaincu dans son utilisation. Utiliser Angular requiert d'être organisé, de vérifier de manière permanente le type de données que nous manipulons et de structurer l'architecture de son code. Ceci ne nous est pas apparu comme une contrainte, mais plus comme une opportunité pour développer

une certaine rigueur qui permettrait à l'avenir de mieux maintenir le code et de passer plus facilement de l'étape du prototype vers celle d'une mise en production.

Relevons-ici que le framework Angular est de moins en moins utilisé. D'après les sondages réalisés en 2023 par le site stackoverflow [27], celui-ci se situe en septième position au classement des technologies web. À ce sujet, Olivier Hondermarck nous explique que par rapport à deux autres frameworks, React et VueJS :

« La popularité du framework a été forte, mais son potentiel de croissance est terminé et il se place en dernière position. » [25]

Cependant, le framework a annoncé une importante mise à jour en novembre 2023 avec Angular 17. Un nouveau site internet a été conçu pour la documentation, puis la syntaxe et de nombreuses fonctionnalités ont été simplifiées [28]. Juste avant le début de ce travail de Bachelor, la mise à jour Angular 18 est arrivée [29]. Ceci nous laisse croire que ce framework dispose d'un avenir encore bien prometteur.

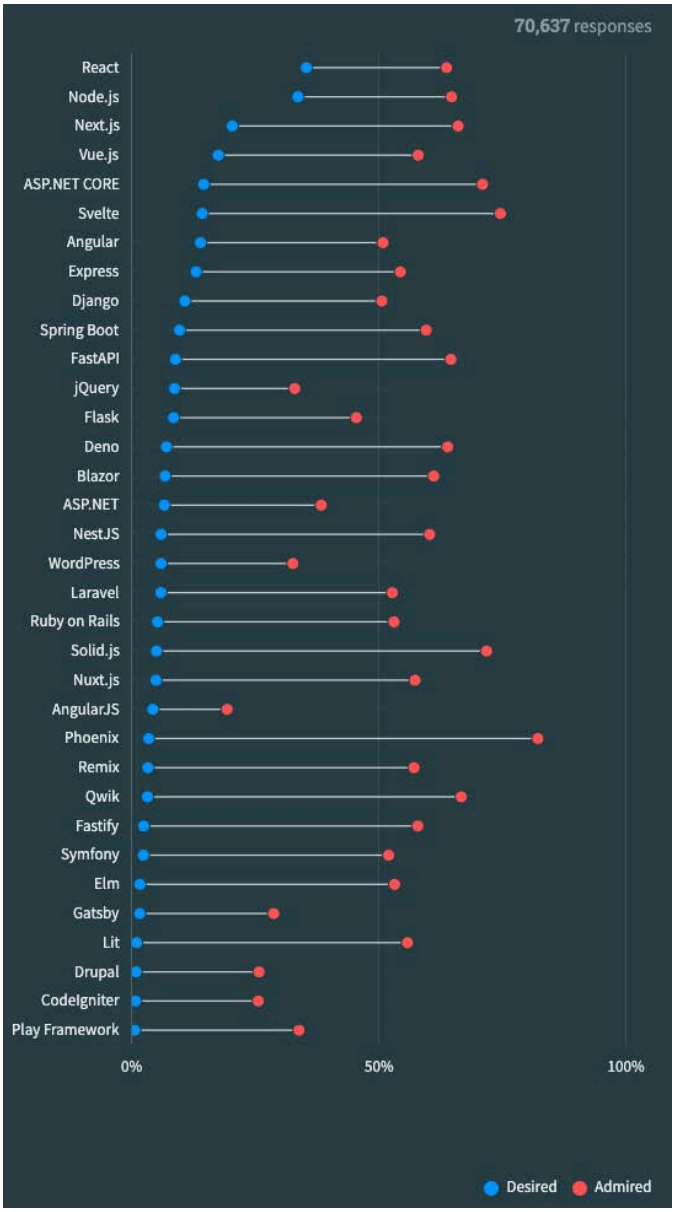

Figure 56 : Sondage concernant les technologies web en 2023

Installation de NodeJS et npm

Pour pouvoir utiliser Angular, il est nécessaire d'installer Node.js qui est un environnement d'exécution JavaScript :

« Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine, which takes JavaScript code and executes it directly on the computer's operating system. It enables developers to run JavaScript code on the server-side, contrary to the traditional approach of executing JavaScript in the browser. » [30]

Il est aussi essentiel d'utiliser npm qui est un gestionnaire de paquets pour Node.js. Grâce à celui-ci, il est possible d'installer facilement l'interface en ligne de commande Angular CLI :

```
1     npm install -g @angular/cli
```

Code 3 : Installation d'Angular CLI avec npm

« The Angular CLI is a tool created by the Angular team that improves the developer experience while building Angular applications. It hides much of the complexity of scaffolding and configuring an Angular application while allowing the developer to concentrate on what they do best - coding! » [26]

Cet outil est en effet bien pratique car il permet de générer un projet Angular, des composants, des services ou encore de « build » notre application avant de la déployer.

Création de l'application Angular

Créer une application Angular peut donc se faire facilement grâce à Angular CLI. Il suffit de se rendre dans le dossier souhaité avec un terminal et d'y entrer la commande suivante :

```
2     ng new my-app
```

Code 4 : Création du projet avec Angular CLI

Dans cette instruction, « my-app » est le nom de l'application. L'ensemble du projet va se générer automatiquement avec les fichiers de configuration liés. Il est possible de connaitre la version installée d'Angular CLI en entrant la commande :

```
3     ng version
```

Code 5 : Instruction pour vérifier la version d'Angular CLI

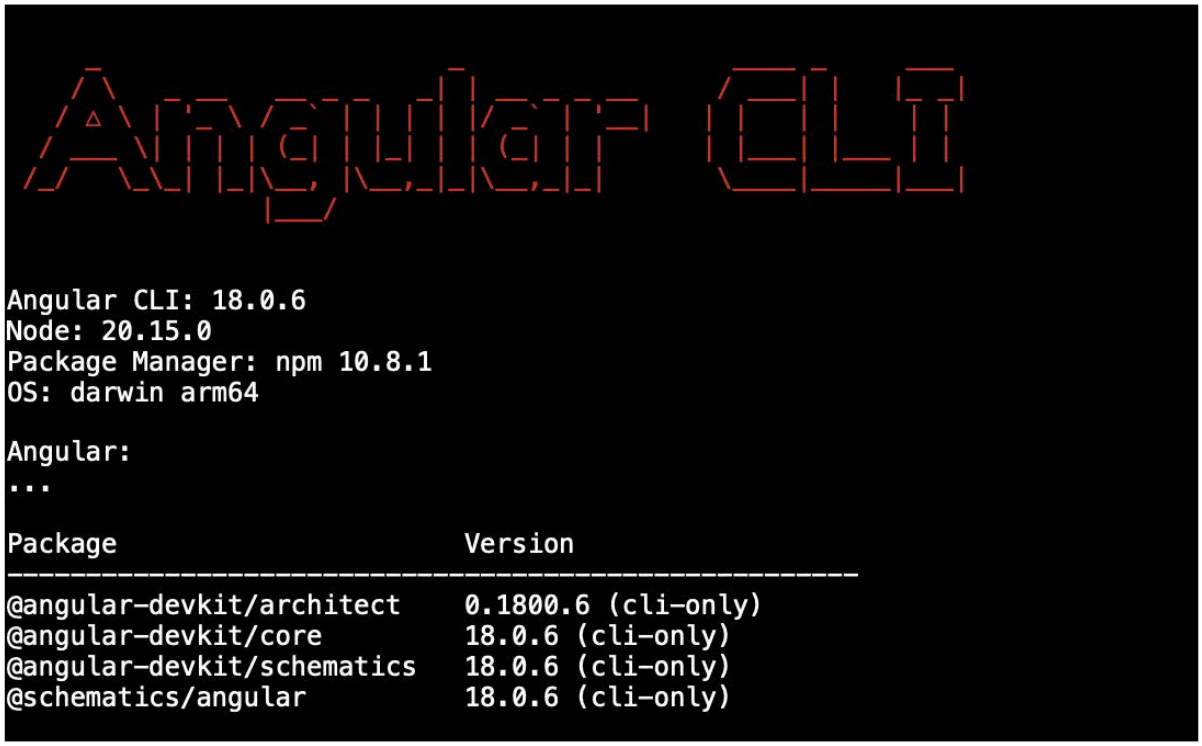

Figure 57 : Versions d'Angular et de Node utilisées pour le prototype le 3 juillet 2024

Concernant les options de configuration demandées par la console lors de la génération du projet, nous avons choisi d'utiliser css pour le format des feuilles de style et de ne pas activer le rendu côté serveur.

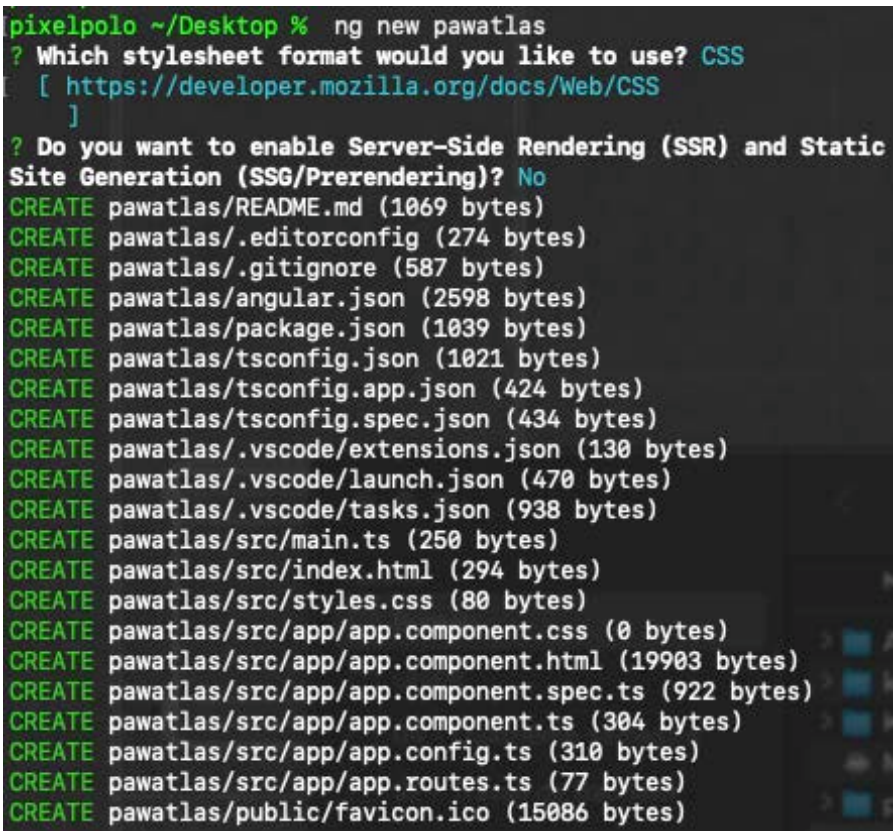

Figure 58 : Option de configuration lors de la génération du projet

Il est important de mentionner que notre application est de type « Single-Page Application ». C'est à dire que l'ensemble du code html, css et JavaScript est téléchargé par le client à partir du serveur Firebase, puis exécuté dans son navigateur. De cette manière, chaque page de notre application est rendue côté client, et non pas côté serveur. Pour plus d'informations concernant cette approche, nous pouvons à nouveau citer les termes d'Aristeidis Bampakos :

« A typical Angular application follows the Single-Page Application (SPA) approach, where each page is created in the DOM of the browser while the user interacts with the application. A web server hosts the application and is responsible for serving only the main page, usually called index.html, at application startup.

Server-Side Rendering (SSR) is a technique that follows an entirely different approach for application rendering than SPA. It uses the server to prerender pages while they are requested at runtime from the user. Rendering content on the server dramatically enhances the performance of a web application and improves its Search Engine Optimization (SEO) capabilities. » [26]

Il est vrai que l'optimisation du référencement aurait été un point important pour notre prototype d'un point de vue marketing. Cependant, l'entreprise E-ProShop Sàrl fera la promotion de notre application à travers un site internet. De plus, activer le rendu côté serveur aurait nécessité de configurer différemment l'hébergement sur Firebase et cette fonctionnalité n'est pas gratuite.

4.2.2 Application web progressive PWA

Après avoir créé l'application comme décrit précédemment, celle-ci comporte la structure classique de tout nouveaux projets générés par Angular CLI. Elle n'est cependant pas encore une application web progressive. Des étapes de configurations supplémentaires sont nécessaires pour la transformer PWA. Ce processus se fait automatiquement par la commande suivante offerte par Angular CLI :

```
1     ng add @angular/pwa
```

Code 6 : Transformation de l'application en PWA

De nombreux éléments se sont produits lors de l'exécution de cette instruction et nous pensons qu'il est intéressant d'analyser en détail ce qu'il s'est passé. Pour mieux comprendre ce processus, commençons tout d'abord par définir ce qu'est exactement une application web progressive PWA. Un excellent livre écrit par Tal Ater nous propose la définition suivante :

« Progressive web apps are a new breed of web apps that combine the benefits of a native app with the low friction of the web. Progressive web apps start off as simple websites, but as the user engages with them, they progressively acquire new powers. They transform from a website into something much more like a traditional, native app. » [31]

De cette définition, il est difficile de comprendre techniquement ce qu'implique une application web progressive. En continuant nos recherches, nous pouvons trouver plus de précisions :

« PWA applications stand between the two worlds of web and native applications and share characteristics from both. A PWA application is a web application that is based on the following pillars to convert into a native one:

- *Capable : It can access locally saved data and interact with peripheral hardware that is connected to the device of the user.*
- *Reliable : It can have the same performance and experience in any network connection, even in areas with low connectivity and coverage.*
- *Installable : It can be installed on the device of the user, can be launched directly from the home screen, and interact with other installed native application. » [26]*

Une application web progressive a donc accès aux données locales du périphérique ainsi qu'à ses composants matériels tel que l'appareil photo ou le GPS. Elle propose une expérience similaire si l'utilisateur est hors ligne et elle peut être installée.

Concernant la connectivité à internet, nous n'avons pas abordé le sujet dans le troisième chapitre de ce dossier lorsque nous avons présenté l'application d'un point de vue utilisateur. En effet, ceci est étroitement liée à l'utilisation de la base de données Firestore que nous allons détailler dans la section concernant le Backend.

Pour mieux comprendre comment une application web peut disposer de ces « capacités », il est nécessaire de s'intéresser à la notion de « service worker ». Ce terme représente un programme qui s'exécute en arrière-plan côté client :

« A service worker is a script that can be registered to control one or more pages of your site. Once installed, a service worker sits outside of any single browser window or tab. From this place, a service worker can listen and act on events from all pages under its control. Events such as requests for files from the web can be intercepted, modified, passed on, and returned to the page. » [31]

De manière plus précise encore, un « service worker » est un programme qui se situe entre les onglets ouverts dans le navigateur du client et le serveur web distant. Avec cette position d'intermédiaire, un peu comme un proxy, il est capable d'intercepter les requêtes http effectuées par le code client pour servir du contenu directement à partir du cache. Il peut à l'inverse communiquer avec le serveur alors que le client a fermé son navigateur, ce qui est pratique pour les notifications push.

Les étapes qui se sont déroulées à la suite de la transformation de notre application en PWA peuvent être listées comme suit :

- Le paquet `@angular/service-worker` a été installé par npm dans le dossier `package.json`. Celui-ci regroupe les dépendances nécessaires pour notre application.
- Un fichier `manifest.webmanifest` a été ajouté dans le dossier `public`. Il contient les propriétés de notre application comme son nom, la couleur de son thème ou encore des icônes qui sont requis pour l'installation. Par défaut ce sont ceux d'Angular qui ont été placés dans le dossier `public/icons`.
- Le fichier `index.html` dans le dossier `src` a été modifié pour y intégrer un lien vers `manifest.webmanifest`. Une balise html définissant des métadonnées a été créée. Elle contient la couleur du thème. Nous l'avons d'ailleurs modifié pour qu'il soit blanc.
- Un fichier `ngsw-config.json` a été ajouté à la racine de notre projet. Celui-ci comprend les configurations nécessaires du « service worker ». Il précise notamment quels fichiers ou données sont à mettre en cache, et de quelle manière.
- Le fichier de configuration `angular.json` a été modifié pour y ajouter `ngsw-config.json` dans les paramètres de « build ».

- Le fichier `app.config.ts` a été modifié pour y ajouter du code qui va permettre d'effectuer l'enregistrement du « service worker » côté client :

```
1     // Service Worker for PWA
2     provideServiceWorker('ngsw-worker.js', {
3       enabled: !isDevMode(),
4       registrationStrategy: 'registerWhenStable:30000',
5     }),
```

Code 7 : Enregistrement du service worker - app.config.ts

Pour plus de détails, nous pouvons nous référer encore une fois au livre écrit par Aristeidis Bampakos à propos d'Angular [26]. Cependant nous utilisons la version 18 du framework. Celle-ci n'utilise plus le concept de modules qui regroupaient les fonctionnalités communes d'une application. Il y a donc quelques différences avec les informations disponibles dans cet ouvrage car il a été rédigé avec la version 12. La figure ci-après nous présente les modifications apportées lors de l'exécution de la commande d'installation de la PWA :

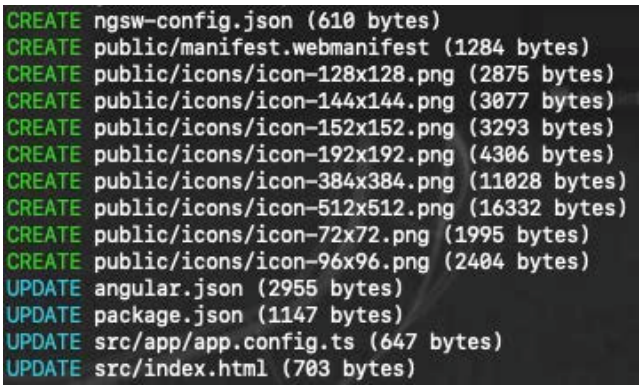

Figure 59 : Modification apportées par ng add @angular/pwa

Pour terminer, nous pouvons ainsi démarrer la construction de notre application grâce à la commande suivante :

```
1     ng build
```

Code 8 : Construction de l'application en vue d'un déploiement

Cette dernière instruction va créer le dossier `dist` qui sera hébergé sur Firebase. De plus, elle va y générer le fichier `ngsw-worker.js` qui contient l'implémentation du « service worker » de notre PWA. Notre application est ainsi prête à être déployée.

4.2.3 Déploiement

Lors de la phase de développement, il est possible d'exécuter la commande suivante pour que notre application soit accessible dans notre navigateur :

```
1     ng serve
```

Code 9 : Construire et servir son application Angular

« You can serve your Angular CLI application with the ng serve command. This will compile your application, skip unnecessary optimizations, start a devserver, and automatically rebuild and live reload any subsequent changes. You can stop the server by pressing Ctrl+C. » [32]

L'application est accessible à l'adresse locale : https://localhost:8080 et la page est automatiquement actualisée lorsque nous apportons des modifications au code.

En revanche, cela n'est pas compatible avec les « services worker ». Pour pouvoir tester pleinement les fonctionnalités de notre prototype, il est nécessaire d'installer le programme http-server [26] ou alors de déployer son application en ligne.

Firebase hosting

Déployer son application avec Firebase hosting est une étape simple et bien documentée [33]. Il est nécessaire de créer un projet dans la console Firebase, puis d'installer sur son ordinateur les outils en ligne de commande Firebase CLI avec l'instruction suivante :

```
1     npm install -g firebase-tools
```

Code 10 : Installation de Firebase CLI

Ensuite, il faut se connecter à l'aide de la commande :

```
2     firebase login
```

Code 11 : Connexion à Firebase

Et enfin, démarrer le processus d'hébergement :

```
3     firebase init
```

Code 12 : Initialisation du processus d'hébergement

Nous sommes invités à répondre à quelques questions, et c'est pour cette raison que nous avons décidé de préciser ces étapes. Il est nécessaire de porter une attention particulière au choix du dossier qui sera mis en ligne. Celui doit contenir le fichier `index.html`, soit le point d'entrée de notre application. Pour notre prototype, il est situé dans le dossier `dist/pawatlas/browser`.

Une fois ce processus terminé, il suffit d'exécuter les commandes suivantes pour déployer son application afin qu'elle soit disponible en ligne et ceci gratuitement :

```
4     ng build
5     firebase deploy
```

Code 13 : Déploiement de l'application

4.2.4 Implémentation

À ce stade, nous avons expliqué comment créer une application web progressive et comment la déployer en ligne. Nous n'avons pas encore analysé l'implémentation de notre prototype. Si nous décidons de le faire sur l'ensemble du code, ce travail écrit serait beaucoup trop conséquent et peu intéressant pour le lecteur. C'est pourquoi nous souhaitons parcourir

uniquement quelques parties fondamentales comme la structure générale de l'application, le système de routage, la notion de service, ou encore certains composants.

Dans la partie consacrée au Backend, nous verrons aussi d'autres éléments du code, notamment comment notre application se connecte à la base de données Firestore.

Structure

« The architecture of an Angular application is based on a hierarchical representation of components. Components are the fundamental building blocks of an Angular application. They represent and control a particulat portion of a web page called the view. » [26]

Une application Angular est structurée en composants qui sont organisés de manière hiérarchique. Dans notre projet, nous pouvons examiner le fichier `index.html`, qui est le point d'entrée de l'application. Il contient seulement deux balises en son corps. Le composant `app-root` est la racine de notre application, c'est-à-dire le sommet de la hiérarchie :

```
1     <body>
2       <app-root></app-root>
3       <noscript>Please enable JavaScript to continue using this
      application.</noscript>
4     </body>
```

Code 14 : Point d'entrée de l'application - index.html

Un composant Angular peut être généré par la commande suivante :

```
1     ng generate component my-component --inline-template=false
```

Code 15 : Création d'un composant Angular

L'option `--inline-template=false` permet de séparer le contenu html et le script TypeScript d'un composant. De cette manière, quatre fichiers sont générés :

- `component.css`, qui contient le style de notre composant.
- `component.html`, qui contient le contenu affiché à l'utilisateur.
- `component.spec.ts`, utilisé pour les tests unitaires.
- `component.ts`, qui contient la logique du composant.

Concernant le composant racine de notre application, celui-ci comprend une balise html `router-outlet` qui intègre le système de routage d'Angular et qui est remplacée de manière dynamique côté client par nos autres composants. De plus, le menu se situe dans une autre balise afin d'être présent dans l'ensemble de notre prototype.

```
1     <div class="my-app">
2       <div class="content">
3         <router-outlet />
4       </div>
5       <app-menu-footer class="footer" />
6     </div>
```

Code 16 : Contenu de notre composant racine - app.component.ts

Routage

Comme mentionné, notre application est réalisée sous la forme de « Single-Page Application ». Cela ne veut pas dire qu'elle ne contient qu'une seule et unique page, mais que celles-ci sont générées côté client.

« In a single-page app, you change what the user sees by showing or hiding portions of the display that correspond to particular components, rather than going out to the server to get a new page. » [34]

Pour ce faire, nous utilisons dans notre prototype un système de routage inclus dans Angular. Nous avons configuré celui-ci dans le fichier `app.routes.ts`.

D'une part, nous pouvons y définir les différentes routes qui représentent les URLs insérés par l'utilisateur dans son navigateur. Ces routes sont utilisées dans notre application pour que l'ensemble de la navigation se fasse à travers le menu situé en bas de l'écran ou encore à l'aide de boutons et de liens. Ceci permet d'éviter que les utilisateurs doivent insérer ces URLs manuellement. Une route contient un chemin d'accès, un titre et un composant à afficher.

```
1       // login
2       { path: 'login', title: 'Login', component: LoginComponent },
3         ...
4       // danger
5       {
6         path: 'danger',
7         title: 'Dangers',
8         component: DangerComponent,
9         canActivate: [authGuard],
10      },
```

Code 17 : Exemple de configuration de routes - app.routes.ts

D'autre part, nous avons mis en place un « guard » Angular que nous avons nommé `authGuard`. Celui-ci permet d'empêcher l'accès à tout utilisateur non connecté. C'est pourquoi certaines routes contiennent une propriété supplémentaire `canActivate: [authGuard]`. La commande utilisée pour générer un « guard » est la suivante :

```
1     ng generate guard your-guard
```

Code 18 : Génération d'un guard Angular

Il existe de nombreuses fonctionnalités et configurations pour les « guard » Angular [35]. Dans notre cas, nous avons simplement utilisé celle qui permet de configurer une fonction qui retourne `true` si un utilisateur est connecté, ou `false` sinon. Ainsi, tout utilisateur non connecté n'a pas accès au contenu de l'application.

```
1     export const authGuard: CanActivateFn = async (route, state) => {
2       // Services
3       const authService = inject(AuthService);
4
5       // Check if the user is logged in
6       const isLoggedIn = await authService.isLogged();
7       if (isLoggedIn) {
8         return true;
9       } else {
10        return false;
11      }
12    };
```

Code 19 : Configuration d'un guard Angular - auth.guard.ts

Injection de Service

Nous pouvons voir dans le code présenté ci-dessus qu'un service a été injecté dans notre « guard ». Un service est un composant qui encapsule des fonctionnalités ou des données et qui peut être accessibles pas plusieurs composants [36].

```
1     ng generate service my-service
```

Code 20 : Génération d'un service Angular

Les services reposent sur le principe d'injection de dépendances qui est un concept très intéressant dans le framework Angular :

« Angular facilitates the interaction between dependency consumers and dependency providers using an abstraction called Injector. When a dependency is requested, the injector checks its registry to see if there is an instance already available there. If not, a new instance is created and stored in the registry. » [37]

Lorsqu'un composant souhaite utiliser un service, il peut l'injecter. Comme mentionné ci-dessus, Angular va vérifier si une instance est déjà disponible, ou dans le cas contraire, en créer une. Ce principe rejoint un design pattern bien connu appelé « Singleton » :

« Il est important pour certaines classes de n'avoir qu'une seule instance. Alors qu'il peut y avoir de nombreuses imprimantes dans un système, il ne doit y avoir qu'un serveur d'impression. Il ne doit y avoir qu'un fichier système et qu'un seul gestionnaire de fenêtres. Un filtre numérique ne peut avoir qu'un seul convertisseur analogique/digital. Un système de comptabilité sera dédié au service d'une seule entreprise. » [38]

Bien entendu, Angular a réalisé une grande partie du travail pour nous. Nous n'avons pas eu besoin d'implémenter la logique qui garantit une seule instance de notre service. C'est bien en ce point que réside la force d'un framework.

Utiliser ce concept dans notre implémentation apporte plusieurs avantages :

- Plusieurs fonctionnalités peuvent être définies dans un seul service et pourront être réutilisées par plusieurs composants. Si la logique d'une fonctionnalité doit changer, il n'y a pas besoin de le faire dans l'ensemble des composant qui l'utilisent.
- Garantir qu'il n'y ait qu'une seule instance d'un service permet d'alléger les ressources nécessaires lors de l'exécution de notre application, ce qui la rend plus rapide car il y a beaucoup moins d'objets qui sont créés.

```
1     import { Injectable } from '@angular/core';
2     import { Timestamp } from '@angular/fire/firestore';
3     import { getIcon, getSvgUrl } from '../../models/categories';
4
5     @Injectable({
6       providedIn: 'root',
7     })
8     export class DesignService {
9
10      public primaryColor: string = '#087042';
11      public primaryColor2: string = '#20D760';
12      public secondaryColor: string = '#3DDC84';
13      public likeColor: string = '#087042';
14      public dislikeColor: string = '#FF0000';
15      public locationColor: string = '#087042';
16      public selectionBorderColor: string = '#007bff';
17      public selectionBackgroundColor: string = '#f0f8ff';
18      public alertColor: string = '#FFFFCC';
19
20      constructor() {}
21
22      // Get the formatted date from a timestamp
23      public getFormattedDate(timestamp: Timestamp | any): string {
24        let date = timestamp.toDate();
25        // Return date + " - " time with 2 digits
26        return (
27          date.getDate() +
28          '/' +
29          (date.getMonth() + 1) +
30          '/' +
31          date.getFullYear() +
32          ' - ' +
33          ('0' + date.getHours()).slice(-2) +
34          ':' +
35          ('0' + date.getMinutes()).slice(-2)
36        );
37      }
38
39      // Get the icon URL from the marker category
40      public getIconURL(markerCategory: string): string {
41        // From categories.ts
42        return getSvgUrl(markerCategory);
43      }
44
45      // Get the leaflet icon URL from the marker category
46      public getLeafletIconURL(markerCategory: string): L.Icon {
47        // From categories.ts
48        return getIcon(markerCategory);
49      }
50    }
```

Code 21 : Exemple de service utilisé pour le design de l'application - design.service.ts

Intégration de la carte OpenStreetMap

Comme mentionné au début de ce dossier, nous avions pour projet d'utiliser l'API Google Map pour afficher une carte interactive dans notre prototype. Après avoir réalisé quelques essais, nous nous sommes aperçus que cela engendrerait beaucoup trop de frais [39]. Il existe plusieurs alternatives mais notre choix s'est porté sur des technologies open-source, notamment sur OpenStreetMap [17] que nous avons déjà mentionné dans le troisième chapitre de ce dossier.

Afin d'intégrer une carte dans notre application, nous avons utilisé la librairie Leaflet [18]. Celle-ci est très facile d'utilisation et elle est très bien documentée [40]. La création d'une carte dans un composant peut se résumer de la façon suivante.

1. Le template html du composant doit contenir une balise de type `div` pour contenir la carte.
2. Un objet de type `Leaflet.Map` doit être créé et son constructeur prend en argument l'identifiant du conteneur html créé dans l'étape précédente.
3. Un design de carte, soit un objet de type `Leaflet.tileLayer`, doit être configuré avec un URL, un niveau de zoom maximum et crédité de ses auteurs pour des questions de droits d'utilisation. Un « layer » par défaut doit être ajouté à la carte.

De plus, il est possible d'ajouter un objet de type `Leaflet.Control` pour pouvoir sélectionner d'autres designs ou encore ajouter un bouton permettant de se localiser.

```
1     // OpenStreetMap tile layer
2     const OSM = L.tileLayer(
3       'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
4       {
5         maxZoom: 18,
6         attribution:
7           '&copy; <a
      href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>',
8       }
9     );
```

Code 22 : Exemple de layer - map-config.ts

Concernant la géolocalisation, nous avons testé l'utilisation du plugin « locatecontrol » [41], mais les résultats observés n'étaient pas très précis.

Nous avons donc créé notre propre service Angular qui utilise l'API Geolocation [42] de JavaScript. Sa configuration [43] permet une localisation satisfaisante à nos yeux.

Comme un service peut être partagé par d'autres composants, nous l'avons réutilisé dans la page « Dangers » pour calculer la distance entre l'utilisateur et un lieu.

```
1     // Get the user's position
2     public getUserPosition(): Observable<GeoPoint> {
3       return new Observable((observer) => {
4         // Success callback
5         const successCallback = (position: GeolocationPosition) => {
6           let userPosition = new GeoPoint(
7             position.coords.latitude,
8             position.coords.longitude
9           );
10          observer.next(userPosition);
11          observer.complete();
12        };
13
14        // Error callback
15        const errorCallback = (error: GeolocationPositionError) => {
16          console.error('Error getting location', error);
17          observer.error(error);
18        };
19
20        // Options for the geolocation
21        const options: PositionOptions = {
22          enableHighAccuracy: true,
23          timeout: 5000,
24          maximumAge: 0,
25        };
26
27        // Get the current position
28        navigator.geolocation.getCurrentPosition(
29          successCallback,
30          errorCallback,
31          options
32        );
33      });
34    }
```

Code 23 : Fonction de localisation - geolocation.service.ts

Ajouter un lieu

Lorsqu'un utilisateur clique sur la carte, cela déclenche un évènement auquel nous avons attaché un écouteur. Grâce à ceci, nous pouvons récupérer facilement les coordonnées géographiques de l'endroit cliqué pour y faire apparaitre un popup de type `Leaflet.popup`.

```
1     // On click event
2     private onMapClick(): void {
3       this.map.on('click', (e: L.LeafletMouseEvent) => {
4         this.longitude = e.latlng.lng;
5         this.latitude = e.latlng.lat;
6         this.showAddPopup();
7       });
8     }
```

Code 24 : Event listener lors d'un clic sur la carte - map.component.ts

Cette méthode est appelée à la construction du composant, après la création de la carte. Ainsi, il a été possible de modifier le contenu du popup pour y intégrer un bouton.

```
1     // Show a popup to add a marker
2     private showAddPopup(): void {
3       // Create a popup
4       this.addPopup = L.popup().setLatLng([
5         Number(this.latitude),
6         Number(this.longitude),
7       ]);
8       // Create a button for the popup
9       const button = document.createElement('button');
10      button.classList.add('btn', 'btn-success', 'btn-sm');
11      button.textContent = 'Ajouter';
12      // Add the button to the popup
13      this.addPopup.setContent(button).openOn(this.map);
14      // Add an event listener to the button
15      button.addEventListener('click', () => {
16        // Open the form to add a marker
17        this.isAddingMarker = true;
18      });
19    }
```

Code 25 : Personnalisation d'un Leaflet.popup - map.component.ts

Lorsque le bouton a été cliqué, nous pouvons voir à la ligne 17 du code ci-dessus que cela transforme un attribut de classe de type boolean en le définissant sur `true`. Cette classe est celle du composant « map » et cette logique nous permet de modifier son template de manière dynamique. En effet, Angular propose une syntaxe spéciale qui permet d'enrichir un code html classique :

« Extend the HTML vocabulary of your applications with special Angular syntax in your templates. For example, Angular helps you get and set DOM (Document Object Model) values dynamically with features such as built-in template functions, variables, event listening, and data binding. » [44]

Afin de faire apparaitre le formulaire, qui est par ailleurs un autre composant de notre application nommé `app-marker-form`, nous avons utilisé le code suivant :

```
1     @if(isAddingMarker || markerToEdit) {
2     <app-marker-form
3       latitude="{{ this.latitude }}"
4       longitude="{{ this.longitude }}"
5       [markerToEdit]="markerToEdit"
6       (formClosed)="onFormClosed()"
7     ></app-marker-form>
8     }
```

Code 26 : Syntaxe de template d'Angular - map.component.html

Comme nous pouvons le voir à la première ligne, une instruction de contrôle [45] de type `if` a été ajoutée. Celle-ci permet de décider ou non d'afficher le composant `app-marker-form` si l'attribut `isAddingMarker` est définit sur `true`. Nous analyserons la présence `markerToEdit` dans la section qui suit au sujet de la modification d'un lieu.

À la troisième et quatrième ligne, deux propriétés du composant `app-marker-form` sont assignées aux valeurs des attribut `this.latitude` et `this.longitude` de `app-map-component`. La syntaxe utilisée avec la double accolade s'appelle une interpolation [46].

Ces quelques lignes nous démontrent la force du framework Angular et la façon dont le contenu d'une page html est généré ou modifié de manière dynamique. En effet, si la variable `isAddingMarker` est à nouveau définie sur `false`, le composant `app-marker-form` disparait de l'écran. Ceci fonctionne aussi pour les coordonnées géographiques car lorsqu'elles viennent à changer, elles auront automatiquement de nouvelles valeurs dans le composant qui les reçoit.

Modifier un lieu

Nous venons de parcourir des notions importantes offertes par le framework Angular. Dans le code 26, nous avons pu analyser comment du contenu html est rendu de manière dynamique et comment des données sont interpolées à partir de la de la classe TypeScript. Pour aller plus loin, nous pouvons nous intéresser à la communication entre composants.

Nous avons brièvement pu constater, toujours dans le code 26, que la latitude et la longitude sont des propriétés de `app-marker-form`. Celles-ci ont été assignées dans le template html du composant `app-map-component`. D'un point de vue hiérarchique, `app-map-component` est le composant « parent » de `app-marker-form`, son « enfant » auquel il a transmis ces données géographiques.

À la cinquième ligne du même code 26, nous pouvons voir une autre syntaxe où cette fois ce n'est pas une interpolation de texte qui est transmise au composant, mais l'objet `markerToEdit`.

De cette manière, le composant `app-marker-form` a une utilité double :

1. Si l'utilisateur souhaite ajouter un lieu sur la carte, `isAddingMarker` est définit sur `true`, le formulaire apparait et les coordonnées géographiques lui sont transmises.
2. Si l'utilisateur souhaite modifier un lieu sur la carte, `markerToEdit` ne serait pas égal à `null` mais contiendrait un objet. Le formulaire apparait et utilise le lieu transmis pour préremplir les informations à éditer.

En sixième ligne du code 26, le composant `app-map-component` est à l'écoute de l'évènement `formClosed`. La méthode `onFormClosed()` serait appelée dans le cas où celui-ci survient. Cette fois, c'est le parent qui écoute l'enfant, car c'est bien `app-marker-form` qui déclenche `formClosed`.

```
1     // Inputs for lat and long of the clicked point
2     @Input() latitude: string = '';
3     @Input() longitude: string = '';
4
5     // Inputs if we want to edit a marker
6     @Input() markerToEdit: Marker | null = null;
7
8     // Outputs events to the parent that the form is closed
9     @Output() formClosed = new EventEmitter<void>();
```

Code 27 : Communication entre composants - marker-form.component.ts

```
1     // On form closed event
2     public onFormClosed(): void {
3       this.isAddingMarker = false;
4       this.markerToEdit = null;
5       this.map.closePopup(this.addPopup);
6     }
```

Code 28 : Communication entre composants - map.component.ts

La communication entre composants est un principe important et un concept puissant offert par le framework Angular. Il permet non seulement de structurer de manière hiérarchique les différentes vues offertes à l'utilisateur, mais aussi de séparer les fonctionnalités de celles-ci dans un code structuré.

La syntaxe offerte par Angular est une habitude à prendre, mais elle est relativement explicite. Par exemple dans le code 27 qui présente les attributs de classe du composant `app-marker-form`, nous pouvons vite comprendre que `@Input` [47] représente une information entrante alors que `@Output` [48] définit un évènement sortant, soit adressé au parent à l'écoute.

Pour conclure cette section liée à l'implémentation, il est vrai que nous n'avons pas abordé l'ensemble des éléments implémentés pour la partie Frontend de notre prototype. Encore une fois, ce serait beaucoup trop ennuyeux pour le lecteur de ce travail. Nous avons quand même pu aborder des notions fondamentales qui sont réutilisées fréquemment dans le code complet de notre application. De plus, dans la partie suivante consacrée au Backend, nous aurons encore l'occasion de revenir sur le code exécuté côté client.

4.3 Backend

Notre Backend repose sur Firebase qui est un produit de Google permettant aux développeurs de créer, déployer et gérer des applications facilement :

« Firebase is Google's mobile and web application development platform that enables developers to build, improve, and grow their app all from one place. It's integrated with Google Cloud, making it scalable and serverless, and it supports a wide range of development tools and languages. » [49]

Firebase propose des services et ceux que nous avons utilisés sont l'hébergement de notre application (Hosting), le système d'authentification (Authentification), une base de données NoSQL en temps réel (Firestore Database) et le service de stockage (Storage).

4.3.1 Firebase Authentification

Pour garantir une authentification sécurisée dans notre application, le service Firebase Authentification peut se paramétrer directement dans la console Firebase. Afin de mieux comprendre comment nous l'avons implémenté, nous pouvons analyser le processus d'inscription de notre prototype.

Inscription

L'inscription dans notre application se fait à partir d'un formulaire. Tout d'abord, nous avons défini un pattern dans une expression régulière afin de garantir un mot de passe suffisamment robuste :

```
1     // Password pattern for the form
2     // ^ and $ are the start and end of the string, avoiding partial
      matches
3     // (?=.*[0-9]) must contain at least one number
4     // (?=.*[a-zA-Z]) must contain at least one letter
5     // (?=.*[A-Z]) must contain at least one uppercase letter
6     // [a-zA-Z0-9!@#$%^&*] can contain letters, numbers and special
      characters
7     private passwordPattern =
8       '^(?=.*[0-9])(?=.*[a-zA-Z])(?=.*[A-Z])([a-zA-Z0-
      9!@#$%^&*\'"]+)$';
```

Code 29 : Expression régulière pour un mot de passe robuste - register.component.ts

Ce pattern est utilisé dans un formulaire réactif, une autre fonctionnalité offerte par le framework Angular [50].

```
1     password: new FormControl('', [
2      Validators.required,
3      Validators.minLength(6),
4      Validators.pattern(this.passwordPattern),
5     ]),
```

Code 30 : Utilisation du pattern pour le mot de passe - register.component.ts

Une fois le formulaire complété et validé par l'utilisateur, celui-ci déclenche la méthode suivante :

```
1     public registerSubmit() {
2       // Get the values from the form
3       const rawForm = this.registerForm.getRawValue();
4       const email = rawForm.email ?? '';
5       const userName = rawForm.userName ?? '';
6       const password = rawForm.password ?? '';
7       // Call the register method from the AuthService
8       this.subscription.add(
9         this.authService.register(userName, email,
      password).subscribe({
10          next: () => {
11            this.successRegistration();
12          },
13          error: (error) => {
14            this.errorRegistration(error);
15          },
16        })
17      );
18    }
```

Code 31 : Inscription d'un nouvel utilisateur - register.component.ts

Deux points importants sont présents dans le code ci-dessus :

- D'une part, cette méthode utilise un service `authService` qui a été injecté. Nous avons créé ce dernier afin d'extraire du composant la logique de communication avec le service Firebase Authentification.

- La méthode de notre service `register(userName, email, password)` a été appelée à travers une souscription. Ceci repose sur des principes de programmation réactive que nous détaillerons dans une section dédiée ci-après.

Pour clarifier ces éléments, il est nécessaire d'analyser le code exécuté par notre service qui communique avec Firebase.

Service d'authentification

Pour utiliser le service d'authentification de Firebase, il est nécessaire de configurer notre application. Cela se fait dans un navigateur à l'aide de la console Firebase, mais aussi dans notre code client où nous devons transmettre certains éléments de notre projet comme son id et sa clé API. Heureusement, de nombreux articles existent à ce sujet, notamment celui de Gabriel Cournelle qui est très bien rédigé [51].

Une fois ceci réalisé, il est possible d'injecter sans erreur le service `Auth` de Firebase dans toute notre application. D'ailleurs, la même étape est requise pour le service Firestore Database en spécifiant quelques éléments de configuration supplémentaires.

```
1     // Provide the Firebase app
2     provideFirebaseApp(() =>
      initializeApp(environment.firebaseConfig)),
3
4     // Provide the Firebase Auth service
5     provideAuth(() => getAuth()),
6
7     // Firestore service
8     provideFirestore(() =>
9       initializeFirestore(getApp(), {
10        // Offline persistence
11        localCache: persistentLocalCache({
12          // Multiple tab support
13          tabManager: persistentMultipleTabManager(),
14        }),
15      })
16    ),
```

Code 32 : Injections des services Firebase Auth et Firestore - app.config.ts

Programmation réactive

Afin de mieux comprendre notre implémentation, prenons le temps de définir quelques notions concernant la programmation réactive. Une bonne définition de ce paradigme a été formulée par Julio Castro dans le journal communautaire Medium :

« Reactive programming is an approach to programming that is designed to handle asynchronous and event-based programming. It is a programming paradigm that focuses on the propagation of changes through the system. In reactive programming, data flows through the system, and changes to that data are propagated automatically. This allows for highly responsive and scalable applications. » [52]

La programmation réactive permet de gérer des situations asynchrones qui sont très fréquentes avec JavaScript lors d'une communication avec un serveur. Lorsqu'un nouvel utilisateur s'enregistre dans notre application, il est nécessaire de déclencher un processus dans notre code afin de l'inscrire auprès de Firebase. Ce processus est réalisé justement de manière asynchrone, c'est-à-dire qu'il ne bloque pas l'exécution de notre code client.

Afin de mieux imager ce concept, prenons l'exemple où un utilisateur se connecte dans notre application et est redirigé vers la carte interactive. Il n'éprouverait pas une bonne expérience si toute l'application se fige le temps d'obtenir les lieux à partir de notre base de données.

De manière traditionnelle dans le langage JavaScript, ces opérations asynchrones sont réalisées grâce au concept de promesses. Cependant, nous avons fait le choix dans notre implémentation d'utiliser la librairie RxJS qui permet de transformer celles-ci en « observables »,

« RxJS is a library for composing asynchronous and event-based programs by using observable sequences. It provides one core type, the Observable, satellite types (Observer, Schedulers, Subjects) and operators inspired by Array methods (map, filter, reduce, every, etc) to allow handling asynchronous events as collections. » [53]

Cette librairie repose sur deux patterns design bien connus, le premier étant « Observateur » :

« Les objets de base de ce modèle sont le sujet et l'observateur. Un sujet peut avoir un nombre quelconque d'observateurs sous sa dépendance. Tous les objets reçoivent une notification chaque fois que le sujet subit une modification de son état. En réponse, chaque observateur interrogera le sujet sur son état afin d'y adapter le sien propre. » [38]

Pour mieux comprendre comment ce pattern est utilisé dans la librairie RxJS, reprenons le code 31 qui est exécuté lorsqu'un nouvel utilisateur s'inscrit :

- La méthode `register(userName, email, password)` retourne un observable, soit un sujet auquel des observateurs peuvent s'abonner.
- L'appel de cette méthode se fait à travers une souscription. C'est à ce moment que l'observateur s'abonne au sujet. Il reste toutefois nécessaire de se désabonner une fois le composant détruit pour éviter des fuites de mémoire [54].
- La méthode `subscribe` contient une fonction qui détaille le comportement à adopter pour l'observateur lorsqu'il reçoit une notification de la part du sujet. Dans notre cas, nous avons défini aux lignes 10 et 11 que la méthode `successRegistration()` doit être appelée. Dans les lignes 13 et 14, nous avons précisé que la méthode `errorRegistration()` devait être exécutée si une erreur nous était parvenue.

L'autre design pattern présent dans ces principes de programmation réactive est celui appelé « Itérateur ». Son objectif et le suivant :

« Fournit un moyen d'accès séquentiel aux éléments d'un agrégat d'objets, sans mettre à découvert la représentation interne de celui-ci. » [38]

Traditionnellement, un agrégat d'objets est une liste d'éléments que nous accédons de manière séquentielle. Lorsque nous utilisons la librairie RxJS, l'agrégat est un objet de type `Observable` et nous avons accès à des données qui sont émises de manière asynchrone.

Une différence subtile est présente ici. Nous n'effectuons pas une itération sur les données d'un `Observable`, mais nous réagissons à ses émissions, d'où le terme de programmation réactive.

Méthode permettant l'inscription

L'implémentation de la méthode `register(userName, email, password)` est particulière. Elle est disponible dans l'extrait de code 33 et nous pouvons remarquer qu'elle est composée d'une succession de fonctions qui retournent des Observables :

```
1     // Register a new user
2     public register(
3       userName: string,
4       email: string,
5       password: string
6     ): Observable<any> {
7       // Check if the display name already
8       return this.checkDisplayNameExists(userName).pipe(
9         switchMap((displayNameExists) => {
10          if (displayNameExists) {
11            return throwError(() => new Error('auth/displayname-
      already-exists'));
12          }
13          // Check if the email already exists
14          return this.checkEmailExists(email);
15        }),
16        switchMap((emailExists) => {
17          if (emailExists) {
18            return throwError(() => new Error('auth/email-already-in-
      use'));
19          }
20          // Create a new user with email and password
21          return from(
22            createUserWithEmailAndPassword(
23              this.firebaseAuth,
24              email,
25              password
26            ).then((response) => {
27              // Update the user profile with the displayName (not
      done automatically)
28              return updateProfile(response.user, { displayName:
      userName }).then(
29                () => response.user.uid
30              );
31            })
32          );
33        }),
34        switchMap((uid) => {
35          // Post the user to the Firestore database
36          const newUser: User = { uid: uid, email: email,
      displayName: userName };
37          return this.firestoreUserService.postUser(newUser);
38        })
39      );
40    }
```

Code 33 : Inscription d'un nouvel utilisateur - auth.service.ts

- L'opérateur `pipe` [55] permet d'enchaîner des méthodes dites *« pipeable »* sur les objets de type `Observable`.

  « A Pipeable Operator is a function that takes an Observable as its input and returns another Observable. It is a pure operation: the previous Observable stays unmodified. » [56]

- Nous avons utilisé `pipe` en ligne 8 afin de pouvoir appliquer la méthode `switchMap` [57] afin de traiter le nouvel observable émis en retour par `checkDisplayNameExists`.
- Ainsi nous vérifions si l'identifiant existe dans notre base de données, puis nous recommençons le processus pour vérifier si le courriel est déjà utilisé.

- Enfin, nous effectuons l'inscription du nouvel utilisateur auprès de Firebase en transformant une promesse retournée par défaut en observable à l'aide de la méthode `from` en ligne 21.

Ces différentes techniques nous permettent d'enchaîner et de transformer différents observables issus d'opération asynchrones. Nous pouvons ainsi traiter les erreurs, émettre des erreurs personnalisées, transformer des flux de données ou encore y appliquer d'autres opérations de manière réactive.

Relevons que Firebase enregistre les utilisateurs inscrits dans le service Authentification. Par défaut, il sauvegarde uniquement l'adresse électronique de la personne, la méthode d'accès (par exemple les réseaux sociaux), les dates de création ou de dernière connexion, et créée un identifiant unique. Dans une perspective proche où certains utilisateurs disposeront de droits privilégiés, nous avons décidé de sauvegarder les personnes inscrites dans une table dédiée de notre base de données. Ceci est effectué à partir de la ligne 34 du code 33.

4.3.2 Firestore Database

Nous venons de parcourir un aperçu des fonctionnalités implémentées dans notre code client afin de communiquer avec le Backend de notre projet. Nous avons utilisé un service fournit par Firebase pour authentifier de nouveaux utilisateurs, ceci sans avoir besoin de créer un serveur à part entière. Nous avons aussi utilisé la librairie RxJS pour traiter nos opérations asynchrones de manière efficace en utilisant deux patterns design.

Ici, nous analyserons comment nous avons utilisé le service Firestore Database offert par Firebase pour mettre en place une base de données :

« Cloud Firestore est une base de données flexible et évolutive pour le développement mobile, Web et serveur à partir de Firebase et Google Cloud. Comme Firebase Realtime Database, il maintient vos données synchronisées entre les applications clientes via des écouteurs en temps réel et offre une prise en charge hors ligne pour mobile et Web afin que vous puissiez créer des applications réactives qui fonctionnent quelle que soit la latence du réseau ou la connectivité Internet. » [58]

Temps réel

Pour mieux comprendre comment nous avons établi notre connexion entre notre code client et notre base de données, il serait judicieux de commencer par mentionner une fonctionnalité importante offerte par Firestore Database. Celle-ci nous permet d'avoir une visualisation de nos données en temps réel.

« Firestore utilizes web technologies such as WebSockets or Long Polling to establish a persistent connection between the client and the Firestore server. » [59]

Lorsqu'un utilisateur ouvre notre application, il est en connexion directe avec la base de données grâce à des technologies websocket qui sont basées sur le protocole http. Cette connexion est persistante. Grâce à ce processus, par exemple quand un point est ajouté sur la carte interactive de notre prototype, l'ensemble des personnes visualisant la zone le verra apparaitre instantanément.

Pour y parvenir, nous avons tout d'abord dû créer un service qui contient les opérations de lecture et d'écriture classique sur notre base de données. Le code 34 de la page suivante montre les opérations permettant de créer, lire, supprimer et modifier un Marker (un lieu).

Comme nous pouvons le voir à la ligne 29, lorsque nous interrogeons la base de données, un observable contenant une liste de Marker est retourné. Ainsi, chaque fois qu'une donnée est ajoutée ou modifiée, l'observable va émettre une liste mise à jour.

```
1     export class FirestoreMarkerService {
2       // ******************
3       // ***** FIELDS *****
4       // ******************
5
6       // Services
7       private firestore = inject(Firestore);
8
9       // Collection reference to the marker collection
10      private markerCollection = collection(this.firestore,
      'marker');
11
12      // ***********************
13      // ***** CONSTRUCTOR *****
14      // ***********************
15      constructor() {}
16
17      // ********************
18      // ***** METHODS *****
19      // ********************
20
21      // ***** CRUD OPERATIONS - ALL MARKERS *****
22
23      // POST a new marker in the Firestore database
24      public postMarker(marker: Marker): Observable<any> {
25        return from(addDoc(this.markerCollection, marker));
26      }
27
28      // GET all the markers from the Firestore database
29      public getMarkers(): Observable<Marker[]> {
30        console.log('Using firestore service');
31        return collectionData(this.markerCollection, { idField: 'id'
      });
32      }
33
34      // DELETE a marker from the Firestore database
35      public deleteMarker(id: string): Observable<void> {
36        const markerDoc = doc(this.markerCollection, id);
37        return from(deleteDoc(markerDoc));
38      }
39
40      // UPDATE a marker in the Firestore database
41      public updateMarker(id: string, newMarker: Marker):
      Observable<void> {
42        const markerDoc = doc(this.markerCollection, id);
43        return from(setDoc(markerDoc, newMarker));
44      }
45    }
```

Code 34 : Opérations CRUD sur la collection Marker - firestore-marker.service.ts

Utilisation du cache

Lorsque nous avons configuré le service Firestore dans notre application comme présenté dans le code 32, nous avons activé le système de mise en cache [60]. De cette manière, si un utilisateur est hors ligne, il peut continuer à voir les dernières données téléchargées lorsqu'il était connecté à internet. D'ailleurs, il peut toujours ajouter des lieux sur la carte interactive ou les modifier.

En effet, les données téléchargées sont mises en cache, puis utilisées en cas de coupure de connexion. Les transactions auprès de la base de données effectuées hors ligne sont aussi enregistrées dans la mémoire du navigateur. Elles sont ensuite réalisées une fois la connexion rétablie.

Ceci ne peut fonctionner que si nous utilisons la bibliothèque standard offerte par Firestore pour établir une connexion directe avec notre base de données. Il est possible cependant d'effectuer des opérations sur la base de données en utilisant les principes d'une API REST [61] à l'aide de requêtes http classiques. C'est pourquoi nous avons modifié le fichier de configuration de notre « service worker » en y ajoutant des groupes de données à mettre en cache [62].

```
1       "dataGroups": [
2         {
3           "name": "api-firestore-requests",
4           "urls": ["https://firestore.googleapis.com/**"],
5           "cacheConfig": {
6             "maxSize": 100,
7             "maxAge": "1d",
8             "timeout": "3s",
9             "strategy": "performance"
10          }
11        },
12        {
13          "name": "firebase-storage-images",
14          "urls": ["https://firebasestorage.googleapis.com/**"],
15          "cacheConfig": {
16            "maxSize": 100,
17            "maxAge": "1d",
18            "timeout": "3s",
19            "strategy": "performance"
20          }
21        }
22      ]
```

Code 35 : Configuration des données à mettre en cache - ngsw-config.json

Cette configuration sera utile le jour où nous changeons la méthode de connexion entre le code client de notre application et la base de données. Cela peut être aussi nécessaire si nous passons à une autre technologie de Backend. De plus, notre « service worker » intègre ici parfaitement son rôle d'intermédiaire entre notre application et le serveur en interceptant les requêtes http et en utilisant la mémoire cache du navigateur.

NoSQL et la gestion des relations

Dans le code source de notre implémentation, nous avons créé trois services offrant les opérations nécessaires sur les collections Like, User et Marker. Nous avons d'ailleurs pu observer le détail de ce dernier dans l'extrait de code 34.

Utiliser une base de données NoSQL implique que ce soit au développeur de décider de l'organisation de ses données. Pour garder une architecture modulable, nous avons créé un service unique intitulé `firestore-pawatlas`.

Les composants de notre application interagissent uniquement avec ce service. Il gère toute la logique nécessaire aux opérations auprès de la base de données, comme la gestion des relations « many-to-many » [63]. De plus, il est le seul à utiliser les trois autres services mentionnés qui fournissent simplement des opérations de type « CRUD ».

Prenons un exemple concret où un utilisateur supprime un lieu de la carte interactive. Dans une base de données NoSQL, il n'y a pas la possibilité de créer des règles de suppressions en cascade. C'est pourquoi nous devons gérer les situations suivantes :

1. Supprimer un lieu implique la suppression de son image qui ne se trouve pas dans la base de données mais dans le service Firebase Storage.
2. Si nous supprimons un lieu, il faut aussi supprimer l'ensemble des votes associés qui sont sauvegardés dans la collection Like.

Par ailleurs, ce service `firestore-pawatlas` comprend des observables publiques qui contiennent à la fois l'ensemble des lieux ajoutés sur la carte interactive, mais aussi ceux de l'utilisateur actuellement connecté. Cela permet de réduire fortement le nombre de requêtes auprès de notre base de données. En effet, l'instanciation de ce service, qui est pour rappel un « Singleton », provoque l'obtention de l'ensemble des lieux stockés dans Firestore. Ce sont ensuite les composants qui s'abonnent à ces observables pour accéder aux données, les manipuler ou encore les trier.

```
1     // DELETE a marker by the current user
2     // - confirm the deletion
3     // - delete the marker
4     // - delete the marker's image
5     // - delete all associated likes
6     public deleteMarker(marker: Marker): Observable<any> {
7       // Confirm the deletion
8       if (!confirm('Êtes-vous sûr de vouloir supprimer ce lieu ?')) {
9         return of(null);
10      }
11      // Delete the marker
12      return
      this.firestoreMarkerService.deleteMarker(marker.id!).pipe(
13        // Delete the image associated with the marker
14        switchMap(() => {
15          if (marker.image !== 'No image') {
16            return this.fileService.deleteImageFromUrl(marker.image);
17          } else {
18            return of(null);
19          }
20        }),
21        // Delete all likes associated with the marker
22        switchMap(() =>
23
      this.firestoreLikesService.getLikeByMarkerID(marker.id!).pipe(
24            switchMap((likes) => {
25              if (likes.length === 0) {
26                return of(null);
27              }
28              // Create an array of observables for deleting all
      likes
29              const deleteLikesObservables = likes.map((like: Like)
      =>
30                this.firestoreLikesService.deleteLike(like.id!)
31              );
32              // Use forkJoin to wait for all deletions to complete
33              return forkJoin(deleteLikesObservables);
34            })
35          )
36        )
37      );
38    }
```

Code 36 : Suppression d'un lieu - firestore-pawatlas.service.ts

5
Conclusion

5.1 Résultat

L'objectif de ce travail de Bachelor a été de répondre à la demande d'une entreprise en réalisant un prototype d'application web progressive. Nous avons commencé ce travail en analysant un cahier de charges et en sélectionnant les cas d'utilisation qui ont permis de développer non seulement une application installable et fonctionnelle, mais aussi une structure extensible facilement.

Tout d'abord, nous nous sommes intéressés aux possibilités offertes par l'application d'un point de vue utilisateur. Nous nous sommes aperçus que la réalisation finale correspondait à ce qui était prévu et nous sommes très satisfait du résultat.

Ensuite, notre attention s'est portée sur l'implémentation. Nous avons pu découvrir comment créer une application web et comment la transformer en PWA grâce au framework Angular. Cette transformation n'étant pas anodine, nous avons pris le temps d'analyser en détail les modifications apportées lors de ce processus. Nous avons aussi ajusté le « service worker » afin de permettre une utilisation partielle de l'application lorsqu'un utilisateur n'est pas connecté à internet.

Par ailleurs, nous avons développé rapidement ce prototype grâce à des services créés par l'entreprise Google, plus particulièrement ceux de la suite de développement Firebase. En utilisant cette dernière, nous avons pu gagner passablement de temps en évitant de développer notre propre serveur. Toutefois, notre implémentation n'a pas perdue de vue l'idée qu'un jour, il serait possible de devoir changer de Backend, notamment de base de données, ou encore de méthodes de connexion à celle-ci.

De plus, nous avons utilisé des technologies NoSQL. La souplesse offerte par celles-ci a passablement compliqué notre implémentation. Nous pouvons affirmer aujourd'hui que lorsqu'il existe des relations importantes entre nos données, une base de données relationnelle peut simplifier grandement notre développement.

D'un point de vue technique, nous avons pu approfondir les implications liées aux opérations asynchrones et appliquer des notions de programmation réactive pour y faire face. Ces concepts ont permis d'utiliser des design patterns essentiels à la création de logiciel.

Pour finir, notre implémentation peut être améliorée. Débutant avec le framework Angular, de nombreux outils proposés par celui-ci n'ont pas été utilisés. Nous pensons par exemple aux « signaux » [64] que nous n'avons pas utilisés mais qui auront été sûrement très utiles pour afficher du contenu de manière dynamique aux utilisateurs. De plus, nous avons abordé les bases de la librairie RxJS et des optimisations sur le traitement de nos données sont certainement réalisables.

5.2 Améliorations et Perspectives

Comme suggéré, des améliorations sont possibles dans notre code. Bien entendu, le projet présenté dans ce dossier est une « photographie à un instant T » d'un processus qui ne s'est pas encore terminé. À ce stade, l'entreprise qui nous a mandaté est en train d'effectuer des tests quant à l'utilisation de la carte interactive. L'ensemble du design sera revu et de nouvelles fonctionnalités vont sûrement voir le jour prochainement.

Toutefois, des optimisations importantes sont à entreprendre dans un avenir proche :

- Lorsqu'un utilisateur se connecte à l'application, un service va effectuer des requêtes auprès de la base de données afin de récupérer l'ensemble des lieux sauvegardés. Nous nous sommes aperçus que cette implémentation n'est pas concevable sur le long terme. Les services de Firebase facturent les opérations auprès d'une base de données Firestore non pas au nombre de requêtes, mais à la quantité de documents retournés. Ainsi, bien que nous ayons à disposition un quota gratuit de cinquante mille documents par jour en lecture, si nous avons une base de données contenant dix mille lieux, cinq connexions suffiraient à consommer nos crédits offerts. De cette manière, nous devons encore réfléchir à une solution afin de réduire ce chiffre. Par exemple, nous pourrions chercher les lieux qui se trouveraient uniquement dans un certain rayon géographique autour d'un utilisateur. Ceci ne serait d'ailleurs pas très complexe à réaliser car nous avons déjà mis en place un service de géolocalisation.
- La mise en cache des données récupérées à partir de la base de données Firestore fonctionne correctement grâce aux technologies offertes par ce service. Ce processus est automatisé en utilisant les outils de connexion par défaut. Cependant, il ne s'applique pas pour les images qui utilisent le service Firebase Storage. Nous avons donc désactivé la possibilité d'ajouter ou de modifier l'image d'un lieu pour un utilisateur étant hors ligne. Afin de rendre possible ce cas d'utilisation, il sera nécessaire d'implémenter une base de données locale au navigateur afin de pouvoir stocker temporairement les images ajoutées. Ceci peut se faire avec des technologies comme IndexedDB [65].
- Concernant les types de données manipulées, nous avons définit des interfaces TypeScript afin de vérifier celles-ci. Cela nous permet par exemple de nous assurer qu'un lieu contient toujours des coordonnées géographiques. De plus, le schéma de notre base de données nous parait être un classique des bases de données relationnelles. C'est pourquoi nous pensons qu'une migration vers SQL peut être une bonne idée, d'autant plus que cela va améliorer l'efficacité de notre application grâce à une optimisation des requêtes avec des jointures.

De cette manière, encore de nombreuses heures de développement sont nécessaires pour arriver à une application efficace et prête pour une mise en production. Le travail décrit dans ce dossier retrace l'expérience réalisée pour l'implémentation d'un prototype. Ce chemin a été très enrichissant et nous a permis de découvrir de nombreuses technologies reliées au web. Nous espérons que la lecture de ce dossier aura été tout autant enrichissante.

A

Code Source

Le code source du projet sera distribué avec plaisir à celles et ceux qui en feront la demande par mail à l'adresse suivante : ricci.paul.p@gmail.com

B

Licence de la Documentation

Ce document est basé sur un modèle créé par Patrik FUHRER et Pedro DE ALMEIDA (Software Engineering Group, University of Fribourg, Switzerland).

Références

[1] «Registre du commerce - E-ProShop Sàrl,» [En ligne]. Available: https://www.moneyhouse.ch/fr/company/e-proshop-sarl-20509652211. [Accès le 30 Juillet 2024].

[2] «Miel de Manuka,» [En ligne]. Available: https://miel-de-manuka.ch/blogs/infos/le-miel-de-manuka-un-tresor-de-la-nature-revele-par-le-dr-peter-molan. [Accès le 30 Juillet 2024].

[3] «Wikipedia - Content management system,» [En ligne]. Available: https://en.wikipedia.org/wiki/Content_management_system. [Accès le 13 Août 2024].

[4] «Shopify,» [En ligne]. Available: https://www.shopify.com/. [Accès le 30 Juillet 2024].

[5] C. Frédéric, «Conseils Marketing,» [En ligne]. Available: https://www.conseilsmarketing.com/promotion-des-ventes/le-mvp-le-minimum-viable-product-produit-minimum-viable/. [Accès le 30 Juillet 2024].

[6] «Console Firebase,» [En ligne]. Available: https://console.firebase.google.com/. [Accès le 10 Août 2024].

[7] W. Lena, «Advanced Data Management,» Berlin, De Gruyter, 2015, pp. 7-8.

[8] B. Rudi, «Les bases de données NoSQL, comprendre et mettre en oeuvre,» Paris, Eyrolles, 2013, p. 91.

[9] C. Peter, «The Entity-Relationship Model - Toward a Unified View of Data,» *ACM Transactions on Database Systems,* pp. 9-10, 1976.

[10] «Bootstrap CSS,» [En ligne]. Available: https://getbootstrap.com/. [Accès le 1 Août 2024].

[11] «Fontawesome,» [En ligne]. Available: https://fontawesome.com/. [Accès le 1 Août 2024].

[12] «Pexels,» [En ligne]. Available: https://www.pexels.com/. [Accès le 1 Août 2024].

[13] K. Oleksandr, «A way to show a prompt to install your Angular PWA both on Android and iOS devices,» 30 Novembre 2019. [En ligne]. Available: https://medium.com/@oleksandr_k/a-way-to-show-a-prompt-to-install-your-angular-pwa-both-on-android-and-ios-devices-7a770f55c54. [Accès le 2 Août 2024].

[14] «Authentifier à l'aide d'Apple avec JavaScript,» [En ligne]. Available: https://firebase.google.com/docs/auth/web/apple?hl=fr. [Accès le 2 Août 2024].

[15] «Authentifiez-vous à l'aide de la connexion Facebook avec JavaScript,» [En ligne]. Available: https://firebase.google.com/docs/auth/web/facebook-login?hl=fr. [Accès le 2 Août 2024].

[16] «Persistance de l'état d'authentification,» [En ligne]. Available: https://firebase.google.com/docs/auth/web/auth-state-persistence?hl=fr. [Accès le 2 Août 2024].

[17] «OpenStreetMap - Copyright,» [En ligne]. Available: https://www.openstreetmap.org/copyright. [Accès le 2 Août 2024].

[18] «Leaflet,» [En ligne]. Available: https://leafletjs.com/. [Accès le 2 Août 2024].

[19] «Featured tile layers,» [En ligne]. Available: https://wiki.openstreetmap.org/wiki/Featured_tile_layers. [Accès le 2 Août 2024].

[20] «Nominatim,» [En ligne]. Available: https://nominatim.org/. [Accès le 2 Août 2024].

[21] «Nominatim Usage Policy,» [En ligne]. Available: https://operations.osmfoundation.org/policies/nominatim/. [Accès le 2 Août 2024].

[22] «Google Map Places API,» [En ligne]. Available: https://developers.google.com/maps/documentation/places/web-service?hl=fr. [Accès le 2 Août 2024].

[23] «Firebase Pricing,» [En ligne]. Available: https://firebase.google.com/pricing. [Accès le 4 Août 2024].

[24] «Firebase - Introduction Geekforgeeks,» [En ligne]. Available: https://www.geeksforgeeks.org/firebase-introduction/. [Accès le 2024 Août 4].

[25] H. Olivier, «Tout JavaScript,» Malakoff, Dunod, 2020, pp. 327-349.

[26] B. Aristeidis, Angular Projects Second Edition, Birmingham: Packt Publishing Ltd, 2021, p. 1.

[27] «Stackoverflow Developer Survey,» 2023. [En ligne]. Available: https://survey.stackoverflow.co/2023/#section-admired-and-desired-web-frameworks-and-technologies. [Accès le Août 4 2024].

[28] «Angular 17 New Features for Modern Web Development!,» 9 Novembre 2023. [En ligne]. Available: https://www.angularminds.com/blog/angular-17-new-features. [Accès le 4 Août 2024].

[29] «What's new angular 18 features and updates?,» 29 Mai 2024. [En ligne]. Available: https://medium.com/@angularminds/whats-new-angular-18-features-and-updates-1142b3be03ff. [Accès le 4 Août 2024].

[30] «What is Node.js and How it Work?,» 21 Décembre 2021. [En ligne]. Available: https://medium.com/@asiandigitalhub/what-is-node-js-and-how-it-work-490f5ecba665. [Accès le 4 Août 2024].

[31] A. Tal, Building Progressive Web Apps, Sebastopol: O'Reilly, 2017.

[32] «Angular CLI - ng serve,» [En ligne]. Available: https://angular.dev/cli/serve. [Accès le 4 Août 2024].

[33] «Hébergement Firebase,» [En ligne]. Available: https://firebase.google.com/docs/hosting?hl=fr. [Accès le 4 Août 2024].

[34] «Angular Routing,» [En ligne]. Available: https://angular.dev/guide/routing. [Accès le 4 Août 2024].

[35] «Angular - Common router tasks,» [En ligne]. Available: https://angular.dev/guide/routing/common-router-tasks#preventing-unauthorized-access. [Accès le 4 Août 2024].

[36] «Angular - Creating an injectable service,» [En ligne]. Available: https://angular.dev/guide/di/creating-injectable-service. [Accès le 4 Août 2024].

[37] «Angular - Understanding dependency injection,» [En ligne]. Available: https://angular.dev/guide/di/dependency-injection. [Accès le 4 Août 2024].

[38] H. R. J. R. V. J. Gamma Erich, Design Patterns, Catalogue de modèles de conception réutilisables, Paris: Vuibert, 2013.

[39] «Google Maps Pricing,» [En ligne]. Available: https://mapsplatform.google.com/pricing/. [Accès le 4 Août 2024].

[40] «Leaflet Tutorials,» [En ligne]. Available: https://leafletjs.com/examples.html. [Accès le 4 Août 2024].

[41] «Leaflet-locatecontrol,» [En ligne]. Available: https://github.com/domoritz/leaflet-locatecontrol?tab=readme-ov-file. [Accès le 4 Août 2024].

[42] «Geolocation API,» [En ligne]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API. [Accès le 4 Août 2024].

[43] G. Michal, «How to get an accurate position estimate from the Geolocation API in JavaScript,» 27 Janvier 2022. [En ligne]. Available: https://dev.to/3dayweek/how-to-get-an-accurate-position-estimate-from-the-geolocation-api-in-javascript-1njf. [Accès le 4 Août 2024].

[44] «Angular - Template Syntax,» [En ligne]. Available: https://angular.dev/guide/templates. [Accès le 4 Août 2024].

[45] «Angular - Built-in control flow,» [En ligne]. Available: https://angular.dev/guide/templates/control-flow. [Accès le 4 Août 2024].

[46] «Angular - Displaying values with interpolation,» [En ligne]. Available: https://angular.dev/guide/templates/interpolation. [Accès le 4 Août 2024].

[47] «Angular - Accepting data with input properties,» [En ligne]. Available: https://angular.dev/guide/components/inputs. [Accès le 4 Août 2024].

[48] «Angular - Custom events with outputs,» [En ligne]. Available: https://angular.dev/guide/components/outputs. [Accès le 4 Août 2024].

[49] «What is Firebase? Undestanding Google's Comprehensive App Development Platform,» [En ligne]. Available: https://medium.com/@future_fanatic/what-is-firebase-understanding-googles-comprehensive-app-development-platform-769b4f7d2822. [Accès le 4 Août 2024].

[50] «Angular - Reactive forms,» [En ligne]. Available: https://angular.dev/guide/forms/reactive-forms. [Accès le 4 Août 2024].

[51] C. Gabrielle, «Firebase authentication in angular,» 6 Janvier 2024. [En ligne]. Available: https://medium.com/@gabriel.cournelle/firebase-authentication-in-angular-ab1b66d041dc. [Accès le 4 Août 2024].

[52] C. Julio, «What is reactive programming?,» 9 Mars 2023. [En ligne]. Available: https://medium.com/@castrojulio/what-is-reactive-programming-4598df1b5c02. [Accès le 4 Août 2024].

[53] «RxJS - Introduction,» [En ligne]. Available: https://rxjs.dev/guide/overview. [Accès le 4 Août 2024].

[54] N. Chidume, «6 Ways to Unsubscribte from Observables in Angular,» 16 Janvier 2020. [En ligne]. Available: https://blog.bitsrc.io/6-ways-to-unsubscribe-from-observables-in-angular-ab912819a78f. [Accès le 4 Août 2024].

[55] B.-V. Lucas, «RxJS - La méthgode pipe(),» 9 Septembre 2022. [En ligne]. Available: https://blog.codewise.fr/rxjs-methode-pipe. [Accès le 4 Août 2024].

[56] «RxJS Operators,» [En ligne]. Available: https://rxjs.dev/guide/operators. [Accès le 4 Août 2024].

[57] «RxJS - switchMap,» [En ligne]. Available: https://www.learnrxjs.io/learn-rxjs/operators/transformation/switchmap. [Accès le 4 Août 2024].

[58] «Cloud Firestore,» [En ligne]. Available: https://firebase.google.com/docs/firestore?hl=fr. [Accès le 4 Août 2024].

[59] K. Sourabh, «Sync in a Blink : Exploring the Magic of Real-Time Data Updates in Firestore,» 8 Juin 2023. [En ligne]. Available: https://hpkaushik121.medium.com/sync-in-a-blink-exploring-the-magic-of-real-time-data-updates-in-firestore-aab30bfe0589. [Accès le 4 Août 2024].

[60] «Accéder aux données hors ligne,» [En ligne]. Available: https://firebase.google.com/docs/firestore/manage-data/enable-offline?hl=fr. [Accès le 4 Août 2024].

[61] «Utiliser l'API REST de Cloud Firestore,» [En ligne]. Available: https://firebase.google.com/docs/firestore/use-rest-api?hl=fr. [Accès le 4 Août 2024].

[62] S. Maxim, «A new Angular Service Worker - creating automatic progressive web apps. Part 1: theory,» 2 Octobre 2017. [En ligne]. Available: https://medium.com/progressive-web-apps/a-new-angular-service-worker-creating-automatic-progressive-web-apps-part-1-theory-37d7d7647cc7. [Accès le 4 Août 2024].

[63] C. Louis, «How to model a many-to-many relationship in Firestore,» 11 Mars 2021. [En ligne]. Available: https://medium.com/firebase-tips-tricks/how-to-secure-many-to-many-relationships-in-firestore-d19f972fd4d3. [Accès le 4 Août 2024].

[64] «Angular - Angular Signals,» [En ligne]. Available: https://angular.dev/guide/signals. [Accès le 13 Août 2024].

[65] «IndexedDB,» [En ligne]. Available: https://developer.mozilla.org/fr/docs/Web/API/IndexedDB_API. [Accès le 13 Août 2024].

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Faculté des sciences économiques et sociales
Wirtschafts- und sozialwissenschaftliche Fakultät
Boulevard de Pérolles 90
CH-1700 Fribourg

D E C L A R A T I O N

Par ma signature, j'atteste avoir rédigé personnellement ce travail écrit et n'avoir utilisé que les sources et moyens autorisés, et mentionné comme telles les citations et paraphrases.

J'ai pris connaissance de la décision du Conseil de Faculté du 09.11.2004 l'autorisant à me retirer le titre conféré sur la base du présent travail dans le cas où ma déclaration ne correspondrait pas à la vérité.

De plus, je déclare que ce travail ou des parties qui le composent, n'ont encore jamais été soumis sous cette forme comme épreuve à valider, conformément à la décision du Conseil de Faculté du 18.11.2013.

Fribourg, le 3 septembre 2024

(signature)