20/12/2017

PAPPL - Mambiance

Rapport de projet

Equipe : Cyril de Catheu, Paola Palmas

Encadrants : Myriam Servières, Vincent Tourre

Sommaire

I. Présentation du projet

1. Contexte

Ce projet s'inscrit dans le cadre du module PAPPL, projet d'application 1, dispensé en option Informatique. Il dure environ trois mois entre le 15 septembre et le 22 décembre 2017.

Il fait suite à deux projets Mambiance datant de 2014 et 2016. Le premier projet a permis la conception d'une application android permettant de caractériser l'ambiance d'un espace urbain au moyen de marqueurs (mots, images, curseurs). L'idée était de permettre à un utilisateur de qualifier l'ambiance qu'il ressentait à un instant donné dans un espace urbain donné, et de pouvoir partager ce ressenti par le biais d'un réseau social. Cependant le premier groupe n'a pas pu mettre en place la base de données sur un serveur distant. Les membres du second projet ont entièrement réécrit le code de l'application et ont ajouté à la saisie d'une ambiance un nouveau marqueur : la rose des ambiances. Finalement, la première équipe a rendu une application construite mais non opérationnelle. La seconde équipe a d'abord modifié le modèle de données de la base de données, et rendu opérationnelle l'application en créant un nouveau code plutôt qu'en reprenant l'ancien.

Le projet Mambiance repose ainsi sur l'association de lieux à des ambiances, décrites par les ressentis et sensations des utilisateurs à l'aide marqueurs proposés par l'application.

2. Besoins

L'application précédente a été rendue fonctionnelle mais quelques problèmes liés à la version d'Android doivent d'abord être résolus. En effet, l'utilisation de l'appareil photo pour saisir une ambiance est impossible, et la géolocalisation de l'appareil ne fonctionne plus. Ensuite, la base de données doit être modifiée, notamment pour prendre un compte plusieurs utilisateurs, et elle devrait être déplacée vers un serveur externe pour permettre aux utilisateurs de pouvoir partager et stocker ces ambiances. En effet, la base de données est actuellement en local sur le téléphone ou la tablette utilisant l'application.

Cette année, le projet consiste donc en particulier à la mise en place d'un serveur et d'une base de données distantes pour l'application. Pour ce faire, quelques étapes ont ainsi été nécessaires comme la modification du modèle de données, la modification de certaines fonctionnalités, et la gestion de comptes utilisateurs.

II. Travail réalisé

1. Vue d'ensemble du projet

Le but de ce projet était de porter l'application sur un serveur distant. Pour ce faire, il y a besoin d'installer une base de données sur un serveur distant. Afin d'assurer la connexion entre l'application android et la base de données, il est nécessaire d'utiliser un Middleware avec lequel l'application interagit via un WebService (cf. image 1).

Les parties base de données et middleware étaient à créer intégralement. De nombreuses modifications sur l'application Android étaient ensuite à réaliser pour s'adapter au nouveau modèle de données et faire communiquer l'application avec le middleware.

Image 1: Schéma explicatif de l'architecture d'un projet Android avec base de données externe

Le Middleware a été codé sous forme d'API REST. Dans un souci de simplicité, les termes Middleware, API et API REST seront dans la suite de ce document confondus.

2. Outils utilisés

a. Outils pour la base de données

Le premier groupe à avoir abordé le projet mambiance avait déjà tenté de réaliser un middleware connecté à une base de données distantes. Ce groupe avait noté que leur solution qui fonctionnait en local ne fonctionnait plus lors du passage à une base de données distantes, et ceci sûrement à cause de réglages serveur impactant le fonctionnement du middleware. Pour éviter d'avoir les mêmes problèmes qui risquaient d'arriver tard dans le projet (et donc laisser peu de temps pour les corriger), nous avons choisi de tout de suite mettre en place un serveur distant.

Le serveur distant a été mis en place grâce au service DigitalOcean. Un serveur sur une machine distante virtuelle Ubuntu Lamp 16.04 est mis en place à l'adresse 95.85.32.82.

C'est sur ce serveur que la base de données et les fichiers pour le Middleware sont contenus.
Le serveur est Apache2. Les modifications des fichiers de configuration du serveur ont été réalisées avec **FileZilla**: une solution de FTP pour se connecter à la machine distante, et **Sublime Text**, un éditeur de texte avancé.

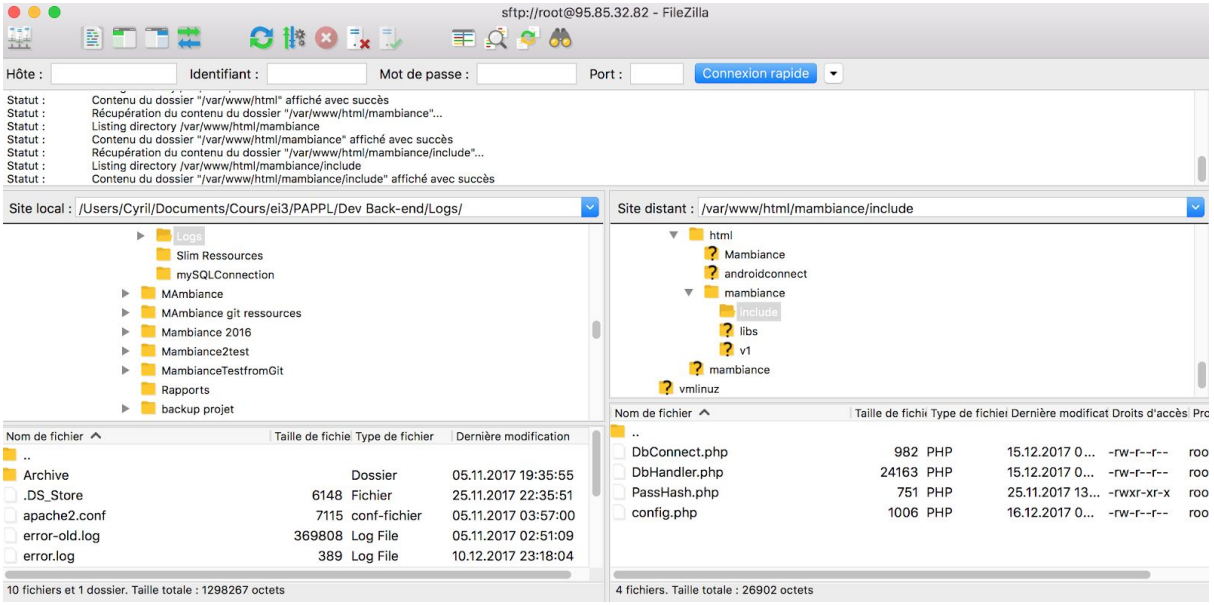


Image 2: FileZilla connecté à la machine distante, vue sur les fichiers du Middleware

La base de données est en **MySQL**. L'administration de la base de données est réalisée avec l'outil **phpMyAdmin**, outil d'administration sous forme d'interface web.

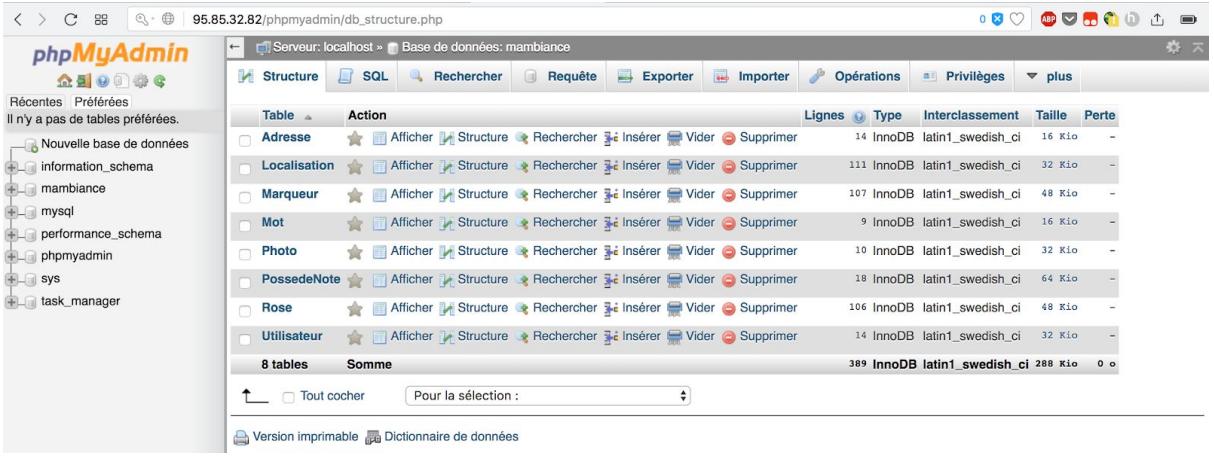


Image 3: Vue de la base de données dans phpMyAdmin

Les configurations détaillées sont fournies en annexes.

b. Outils pour le Middleware

Le Middleware a été codé en **PHP**. Nous n'avons pas utilisé de système de gestion de version. Nous avons connecté l'**IDE Netbeans** directement à la machine distante. Les fichiers modifiés sur Netbeans étaient directement uploadés sur la machine distante.

Cela a facilité le déploiement des modifications et les tests. Néanmoins, il ne serait pas possible de faire cela dans un environnement de production.

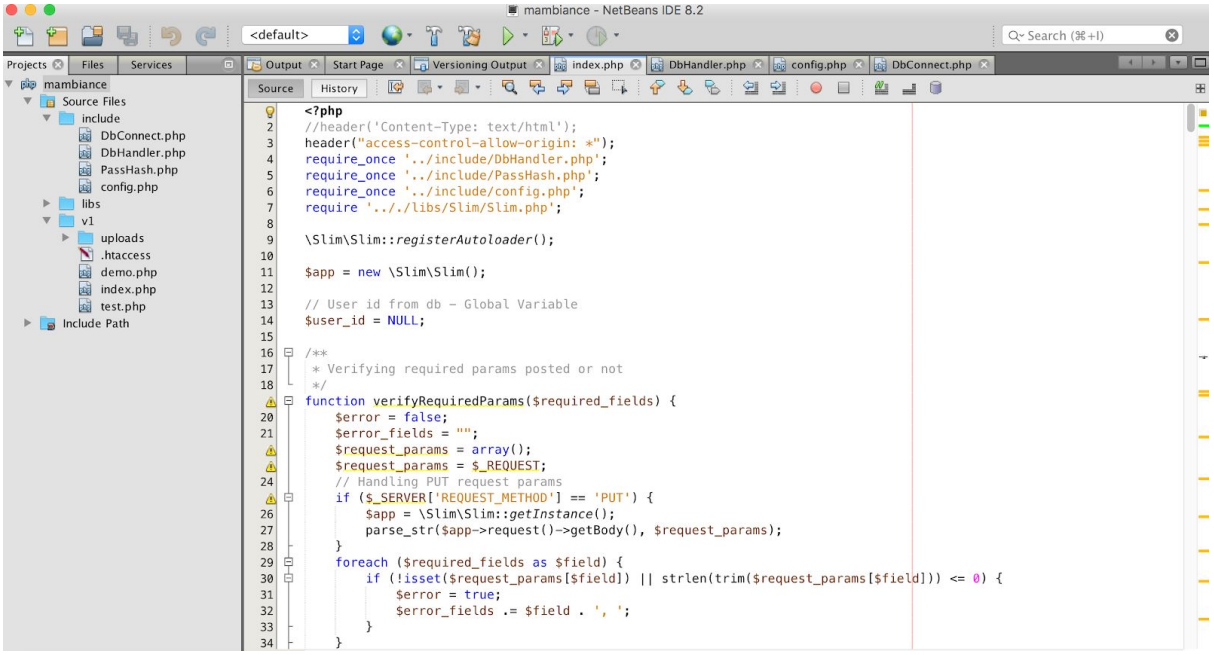


Image 4: Netbeans connecté aux fichiers du Middleware stocké sur la machine distante

Le Middleware a été testé grâce à l'application **Postman.** Postman est une suite de gestion de développement d'API. Cette application a été indispensable à la bonne conduite du projet. Postman permet de créer des requêtes, de les enregistrer, de les documenter, et des les exporter sous forme de code. Les utilisations de Postman sont détaillées dans la partie Tests.

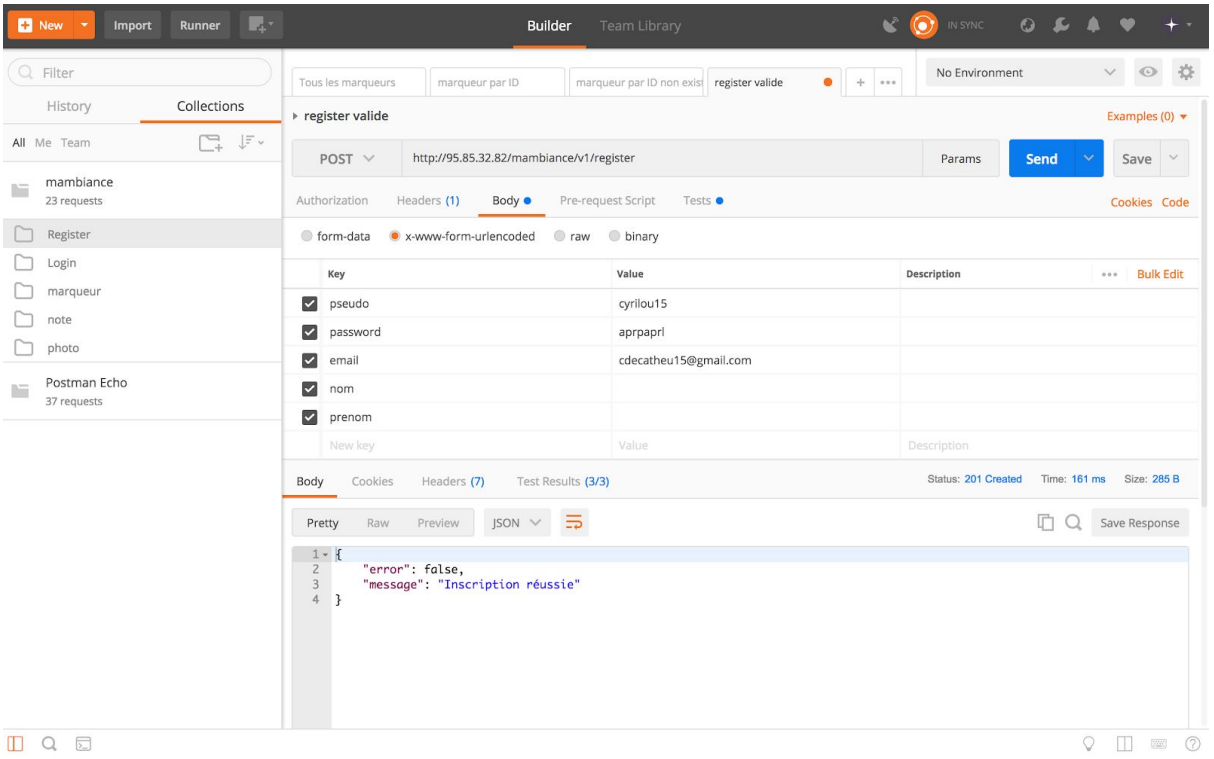


Image 5: Vue d'un test dans Postman

c. Outils pour l'application mobile

Nous avons récupéré le travail fait via **GitHub** au lien suivant :
https://github.com/remiparrot/Mambiance
Puis nous avons créé un nouveau dépôt pour construire plus aisément le projet à deux, et également pour avoir des sauvegardes du travail à différentes étapes et pour le versionning. Le projet est disponible au lien suivant :
https://github.com/ppalmas/MAmbiance

Ensuite, pour développer cette application, nous avons utilisé **AndroidStudio** qui est une IDE qui facilite la production d'applications Android. Il s'agit en fait de l'IDE officiel d'Android. Il possède de nombreuses fonctionnalités intéressantes : l'édition de fichier java, la gestion automatique de la construction de l'application grâce à Gradle, la prévisualisation et l'édition graphique des layout, et bien sûr la compilation et l'installation sur un terminal Android. L'IDE possède aussi un mode debug directement en lien avec le terminal, ce qui est extrêmement pratique et efficace pour corriger les bugs.

Pour gérer la connexion entre Android et le serveur, nous nous sommes appuyés sur le code généré par **Postman** pour gérer les requêtes côté Android.

3. Base de données

Le modèle de données que nous avons construit est largement basé sur le précédent. Nous y avons cependant ajouté les tables Utilisateur et Adresse, et supprimé la table Curseur pour une table PossedeNote.
Le précédent modèle de données et le nouveau sont disponibles ci-dessous.

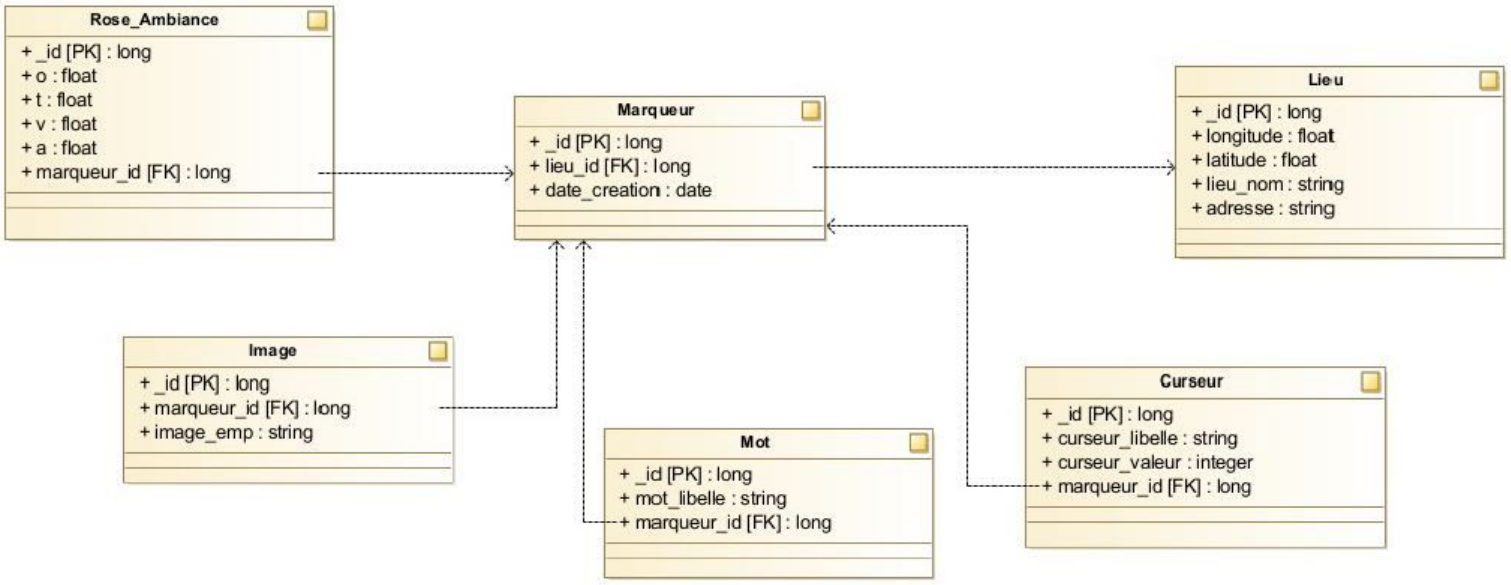


Image 6 : Ancien modèle physique de données

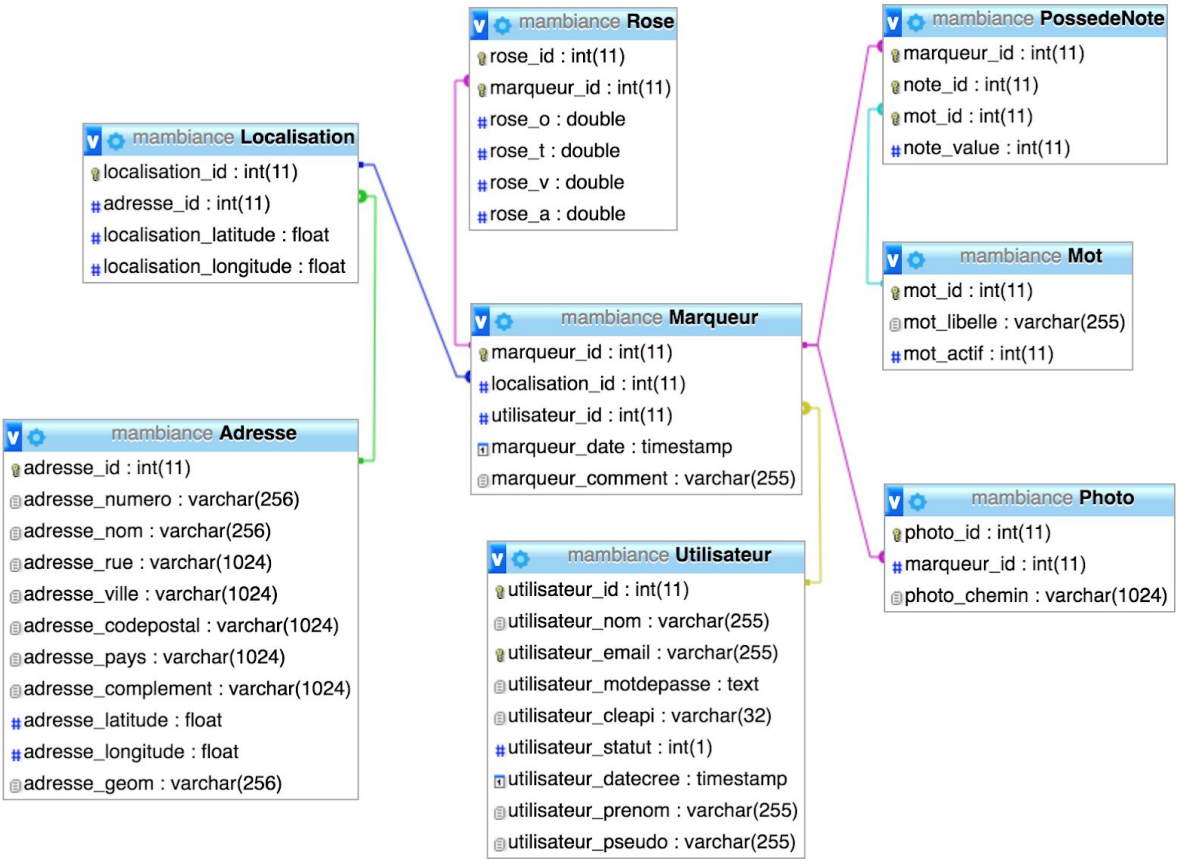


Image 7 : Modèle physique de données actuel

Le modèle de données est ainsi basé sur le précédent, avec quelques modifications explicitées ci-dessous :

Tables Mot et PossedeNote :
La table curseur correspond plus ou moins aux tables Mot et PossedeNote, à ceci près que la table PossedeNote répertorie les notes entrées par les utilisateurs et associées aux mots déjà existants dans la table Mot. En effet auparavant, les mots que l'utilisateur devaient noter n'étaient pas dans la base de données, mais entrés manuellement dans le code Android.
Dans la table Mot, un booléen (0 ou 1) *mot_actif* permet de choisir quel mot sera affiché dans l'application et que l'utilisateur pourra noter. Si le booléen vaut 1, le mot est ajouté dans la base. Ces mots sont ajoutés dans la base de données dès l'installation de l'application (cf. fichier *SqliteDataBaseHelper.java*).
En réalité, le booléen est stocké en tant qu'entier. Cela peut permettre d'utiliser cette valeur comme probabilité d'apparition du mot plus fine si le besoin apparaît dans le futur.

Tables Adresse et Lieu :
Par rapport à l'ancien modèle, nous avons ajouté une table Adresse qui permet d'associer le lieu du marqueur à une adresse potentiellement déjà entrée dans la base. Cela permettrait notamment d'associer à un seul lieu emblématique (une place par

exemple), des marqueurs pris à différents endroits de ce même lieu, mais représentant, malgré les différences de coordonnées, le même endroit.

Table Utilisateur :
La table Utilisateur possède plusieurs attributs : nom, prénom, email, pseudo, mot de passe, datecree, une clé api et un statut pour le serveur distant.
L'utilisateur pourra ainsi se connecter soit avec son adresse mail, soit son pseudo.
La clé API sert à ne pas faire de requêtes impliquant l'envoi des identifiants à chaque appel nécessitant d'être authentifié. Elle est un jeton secret pour l'authentification. Utiliser une clé API est un standard de sécurité en gestion de compte utilisateur.
Le mot de passe est stocké sous forme de hash: le mot de passe en clair de l'utilisateur n'est jamais stocké. Dans la base locale, l'utilisateur est enregistré, mais son mot de passe ne l'est tout simplement pas.

Table Marqueur :
Par rapport au précédent modèle de données, on ajoute dans un marqueur, la description du marqueur que l'utilisateur peut entrer lors de la saisie d'une ambiance.

Ces modifications de la base de données ont évidemment induit des changements dans les fichiers permettant de gérer la base de données locale (getters, setters, méthodes employant des requêtes à la base...).

4. Architecture du Middleware

a. Principe

Le middleware est lui même séparé en deux parties:

- une interface qui communique avec les clients : ici, la partie qui communique avec l'application Android
- une interface qui communique avec la base de données

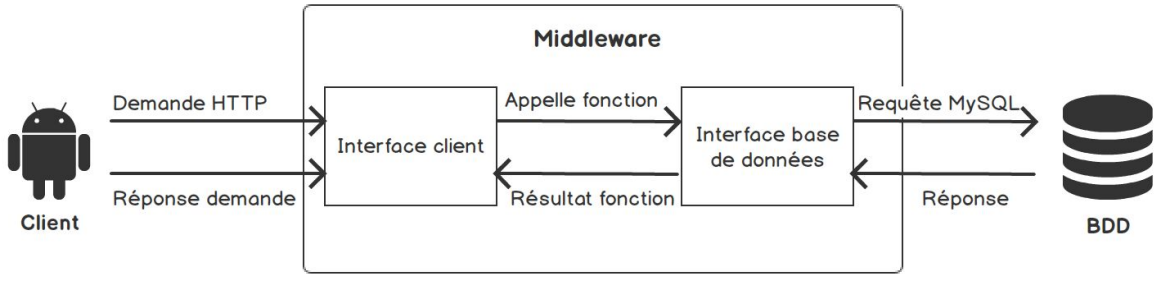


Image 6: Schéma de principe du Middleware

Ce principe permet de ne pas mélanger dans une même fonction la gestion de demande client et la gestion de requête serveur.

Finalement:

- Une fonction de l'interface client prend en argument une requête HTTP et renvoie un statut HTTP (200, 404) et une réponse JSON.
- Une fonction de l'interface base de données prend en argument des variables valuées en PHP et renvoie un stream de données correspondant à une réponse de base de données ou une valeur d'erreur sous forme d'entier.

b. Interface base de données

L'interface base de données est contenue dans un dossier include. Le dossier contient 4 fichiers php:

- Connexion à la base de données : *DbConnect.php*
- Ensemble des fonctions de requêtes à la base de données : *Dbhandler.php*
- Fonction de hash des mots de passe : *PassHash.php*
- Fichier de configuration contenant les identifiants de la base de données : *config.php*

C'est le fichier *DbHandler.php* qui est le plus volumineux. Il contient une fonction pour chaque requête qui doit pouvoir être faite à la base de données.

Principe d'une fonction de *DbHandler.php*:

- ❏ préparation d'une requête MySQL
- ❏ envoi de la requête MySQL
- ❏ gestion et retour en fonction du résultat de la requête

c. Interface client

La gestion de requêtes HTTP nécessite normalement beaucoup de code. La plupart des projets utilisent un framework: librairie de fonctions optimisées permettant la gestion d'opérations classiques plus ou moins complexes.
Nous avons choisi le framework **Slim PHP v2.** Ce n'est pas un framework très puissant, mais il a l'avantage de pouvoir être pris en main rapidement contrairement à des frameworks plus complets et connus comme Laravel ou Symphony.
L'installation du framework est contenue dans le middleware plutôt qu'au niveau du serveur. Cela évitera des problèmes lors de la mise en place sur un serveur de Centrale Nantes. Les fichiers du framework sont dans un dossier libs.

L'interface client est contenue dans un dossier v1. Le dossier contient 2 fichiers et 2 dossiers :

- ❏ Dossier technique qui contient temporairement les images uploadées : uploads-temp
- ❏ Dossier qui contient les images uploadée : uploads
- ❏ Fichier de configuration des accès: *.htaccess*
- ❏ Fichier principal de gestion des requêtes HTTP : *index.php*

Le principe d'une fonction de *index.php* est détaillé ci-dessous :

- ❏ Vérification de l'existence des arguments obligatoires dans la requête HTTP
- ❏ Récupération des arguments de la requête HTTP
- ❏ Appel de fonctions de l'interface base de données
- ❏ Mise en forme du résultat des fonctions
- ❏ Envoie de la réponse au client en JSON

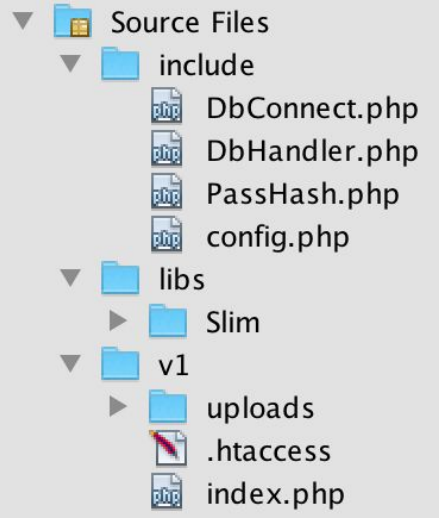


Image 7: Arborescence des fichiers du Middleware
Noter que le dossier uploads-temp n'est pas visible car caché

5. Fonctionnalités du Middleware

Le Middleware décrit ci-dessus permet donc d'accepter des requêtes HTTP pour créer et lire des données dans la base distante. Le client de ce Middleware peut être une application Android comme une application web ou iOS.
Les requêtes disponibles sont :

- ➔ Création d'un utilisateur
- ➔ Connexion d'un utilisateur
- ➔ Création d'un marqueur
- ➔ Obtention de tous les marqueurs et des photos associées
- ➔ Obtention de tous les marqueurs de l'utilisateur
- ➔ Obtention d'un marqueur par son ID
- ➔ Création d'une note
- ➔ Obtention de toutes les notes d'un marqueur par son ID
- ➔ Création d'une photo

Pour plus de lisibilité, toutes les différentes requêtes sont présentées sous forme de documentation web à l'adresse suivante :
https://documenter.getpostman.com/view/3191727/mambiance-documentation/7LkfiBj

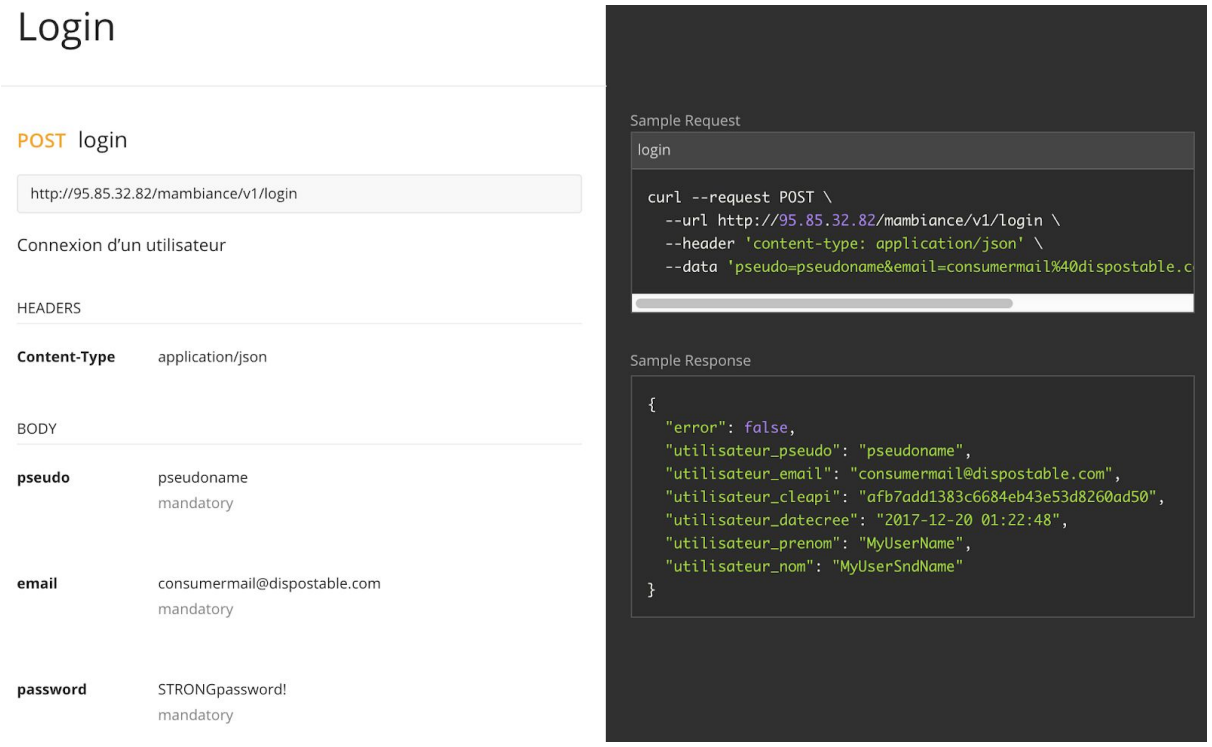


Image 8: Documentation de la requête de Connexion d'un utilisateur

On peut voir en bas à droite de l'image la réponse du middleware à une requête. Néanmoins, les résultats d'une requête sont normalement directement utilisés par le client, ici l'application Android, qui les remet en forme dans une interface graphique.

6. Architecture de l'application Android

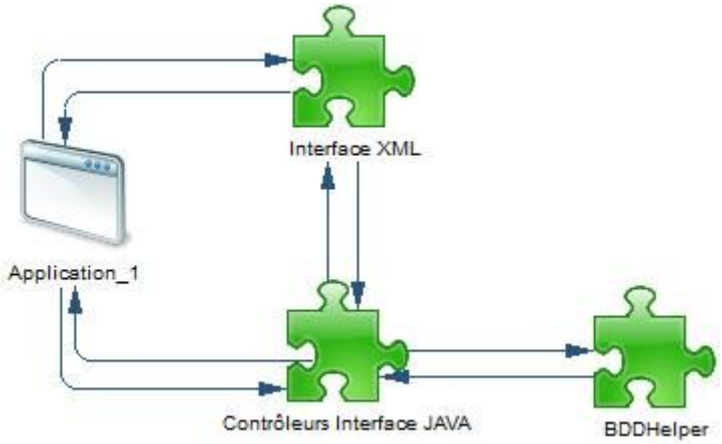


Image 10 : Schéma explicatif de la structure d'un projet Android

L'architecture d'un projet Android sans base de données externe mais avec une base de données locale peut se caractériser simplement par une interface XML, des classes java contrôlant l'interface, le BDDHelper, et l'outil Android utilisé (cf. image 10). Nous allons préciser par la suite l'architecture de l'application Mambiance.

a. Ecrans

L'ensemble des fichiers qui définissent les layout des différents écrans sont localisés dans le dossier res, i.e. ressources, au format de fichiers XML. Ces layout correspondent aux vues de chaque activité, i.e. à l'interface de chaque écran de l'application.
Il y a 8 **layouts** :

- le menu principal : *activity_main.xml*
- l'écran de saisie du marqueur : *edit_activity.xml*
- l'historique : *history_activity.xml*
- la carte : *map_activity.xml* (il s'agit d'ailleurs du layout de deux activités)
- l'affichage d'un marqueur enregistré : *display_marker_activity.xml*
- l'écran d'information : *info_activity.xml*
- l'écran de saisie d'un nouvel utilisateur : *user_activity.xml*
- l'écran de profil de l'utilisateur : *user_profile_activity.xml*

Il y a en outre trois layouts supplémentaires qui permettent d'afficher la liste des marqueurs dans l'historique (*list_item_history.xml*), la rose des ambiances (*rose_ambiance.xml*) et d'afficher la pop-up de choix d'une adresse de localisation du marqueur (*popup_address.xml*).

Par ailleurs, deux fichiers XML définissent la barre de **menu** que l'on voit en haut de l'écran :

- la barre de menu principale : *main_menu.xml*
- la barre de menu des cartes : *menu_map.xml*

b. Activités

Les activités sont la structure de base d'une application Android, elles correspondent à des classes java qui héritent de la classe Activity (et sont stockées dans le dossiers src). Elles contiennent le comportement interne de chaque écran (backend), et elles sont liées à un layout (dans res) pour permettre leur affichage. L'application est ainsi composée de 9 écrans:

- le menu principal : *MainActivity.java*
- l'écran de saisie du marqueur : *EditActivity.java*
- l'historique : *HistoryActivity.java*
- la carte des marqueurs de l'appareil : *MapMarkerActivity.java*
- la carte de tous les marqueurs du serveur: *MapMarkerActivityAll.java*
- l'affichage d'un marqueur enregistré dans l'appareil : *DisplayMarkerActivity.java*
- l'écran d'information : *InfoActivity.java*
- l'écran d'inscription d'un utilisateur : *UserNewActivity.java*
- l'écran de profil de l'utilisateur : *UserActivity.java*

Certaines classes supplémentaires sont nécessaires pour gérer divers outils :

- les marqueurs sur les cartes : *MarkerIconOverlay.java*
- la rose des ambiances : *RoseSurfaceView.java*
- les barres de notation utilisées pour noter une ambiance : *VerticalSeekBar.java*

c. Classes d'échange avec la base de données interne

L'échange avec la base de données interne en SQLite (SGBD sous Android) nécessite l'utilisation de plusieurs classes java (stockées dans un sous-dossier src/db) :

- Une classe par table : Adresse.java, Marqueur.java, RoseAmbiance.java, Image.java, Mot.java, PossedeNote.java, Lieu.java, Utilisateur.java
- Un DAO (Data Access Object), c'est-à-dire une classe contenant des méthodes d'accès à la BDD. Principalement les méthodes CRUD (Create, Read, Update, Delete), mais aussi des méthodes permettant de faire des requêtes plus spécifiques. Le fichier correspondant est LocalDataSource.java. Nous avons repris le fichier existant et complété. A l'avenir, dans le but de clarifier le code, il pourrait être intéressant de séparer le fichier LocalDataSource.java en plusieurs fichiers DAO - i.e. un par classe.
- Un fichier BDDHelper, une classe héritant de SQLiteOpenHelper qui est utilisé pour les instructions de création et de mise à jour des tables de la base de données. Il contient les noms des tables et colonnes, il faut donc s'y référer pour chaque requête. Ce fichier est appelé à chaque création de la base de données. Le fichier correspondant est MySQLiteHelper.java. A chaque changement de ce fichier, la variable DATABASE_VERSION doit être incrémentée de 1 pour permettre à la base de données locale de l'appareil utilisé de prendre en compte ces changements

d. API OSM et librairies

Pour l'affichage des cartes et des marqueurs dans les fichiers *MapActivity* et *MapActivityAll*, on utilise Open Street Map.

Pour gérer la connexion au serveur distant ainsi que la réception et l'envoi de données vers celui-ci, on utilise la librairie OkHttpClient qui simplifie grandement la mise en place et l'utilisation de requêtes HTTP vers le serveur. Il suffit d'ajouter pour cela, dans le fichier *build.gradle* (*Module: app*), dans la partie *dependencies*, la ligne suivante :
compile **'com.squareup.okhttp3:okhttp:3.9.1'**

7. Interface

L'esthétique de la précédente interface a été conservée. Les couleurs sont blanc et vert, l'application possède un écran d'accueil qui permet de cliquer sur des tuiles qui orientent vers différentes fonctionnalités. Il y a 5 tuiles (cf. image 11) :

- une pour accéder au profil utilisateur ou se connecter
- une pour saisir une ambiance - cette action nécessitant de s'être connecté, si l'utilisateur clique sur cette tuile sans l'être, il est redirigé vers l'écran de connexion
- une pour accéder à l'historique de l'appareil
- une pour accéder à la carte des marqueurs

- une pour accéder aux informations liées à cette application

Un bouton en haut à droite permet à tout moment de revenir au menu.

L'écran *utilisateur* et *nouvel utilisateur* n'existaient pas dans le précédent projet, nous les avons ajoutés.

L'interface n'a pas été beaucoup modifiée. Aussi, l'écran *saisie des ambiances* est toujours relativement inadapté à un affichage vertical sur un smartphone. L'écran *utilisateur* n'a en revanche pas été fait pour un format tablette puisque nous disposions pas d'un tel appareil pour effectuer des tests.

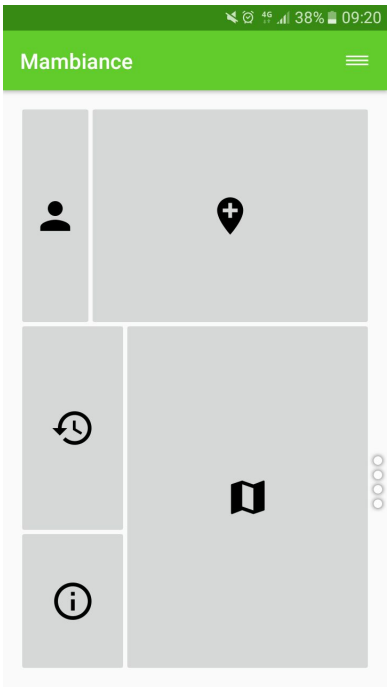


Image 11 : Ecran d'accueil

8. Fonctionnalités de l'application

a. Utilisateur

L'écran utilisateur n'existait dans le précédent projet. Nous l'avons créé pour permettre à différents utilisateurs d'utiliser l'application, en vue de pouvoir envoyer leurs données vers un serveur distant.

Présentation

Lorsqu'il clique sur *saisie des ambiances* ou sur l'icône *utilisateur*, si aucun utilisateur n'est enregistré sur l'appareil, l'utilisateur peut s'inscrire (cf. image 12). Il entre les données suivantes :

- Nom
- Prénom
- Pseudo *
- Email *

- Mot de passe * (et confirmation du mot de passe)

(* champs obligatoires)

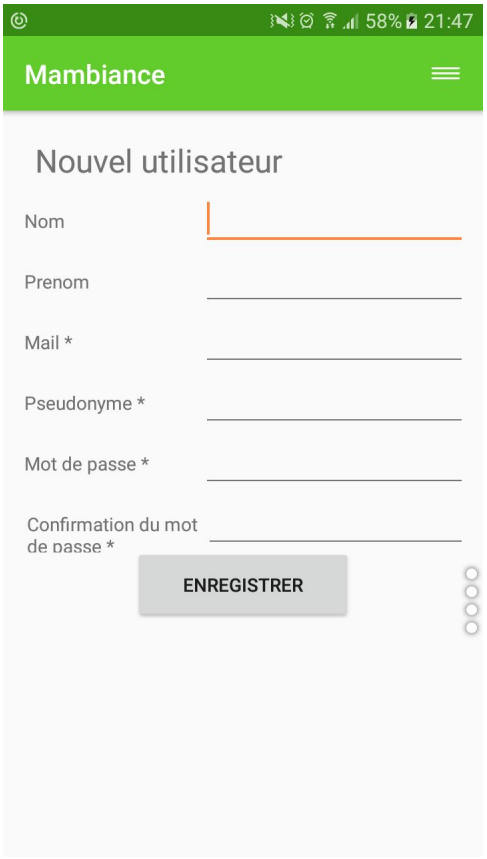


Image 12 : Ecran inscription utilisateur

Une fonction vérifie que l'utilisateur a bien entré les champs obligatoires, que le mot de passe entré fait plus de 6 caractères, et qu'il est identique à la chaîne de caractères entrée dans "Confirmation du mot de passe". Il y a également une vérification effectuée quant à la validité syntaxique de l'email. Si l'email est formé de cette manière: "aa@aa.aa" (aa ensemble de caractères) alors il est valide.

Si un utilisateur est déjà enregistré sur l'appareil, il n'a pas besoin de se connecter et peut consulter son profil en cliquant sur l'icône *utilisateur* (cf. image 13). L'utilisateur est ainsi à la fois enregistré sur la base externe et la base locale. Toutefois, le mot de passe de l'utilisateur n'est pas stocké en local.

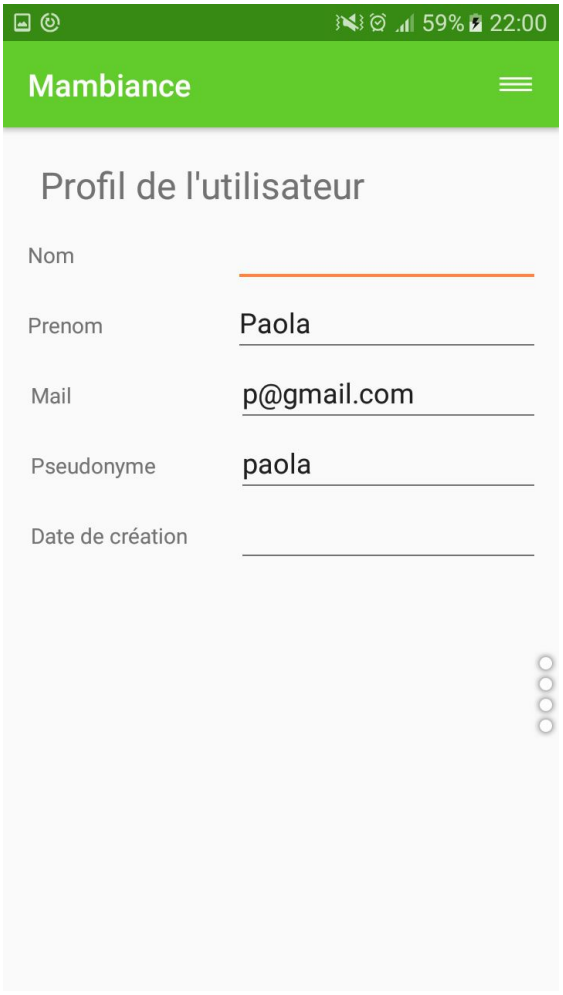


Image 13 : Ecran de profil utilisateur

Points d'amélioration

Avec la configuration actuelle, un utilisateur ne peut se déconnecter ni se connecter sur l'appareil. Le compte utilisateur est ainsi pour le moment lié à l'appareil d'utilisation. En outre, les informations de l'utilisateur - hormis son mot de passe - sont actuellement également enregistrées sur la base de données locale, en plus de la base de données externe.

b. Saisie des ambiances

Saisie de l'ambiance

Le formulaire de saisie d'ambiance a été relativement conservé. Nous n'y avons pas apporté de modification majeure, si ce n'est la partie description. Il ne s'agit plus d'entrer 3 mots, mais plutôt une description générale du lieu.
Par ailleurs, l'utilisateur peut toujours mettre des notes à trois mots choisis aléatoirement dans la table Mot, parmi ceux dont l'attribut *booléen* vaut 1. Un administrateur peut ainsi choisir quels mots l'utilisateur notera en modifiant la base de

données avant utilisation. Toutefois, les mots pris actuellement en compte le sont à partir de la base locale. La base de données distante a donc été synchronisée manuellement (cf. Points d'amélioration).

Enfin, la vérification du formulaire de saisie a été améliorée. Désormais, un message d'erreur ciblé s'affiche. Si par exemple l'utilisateur n'entre pas une description, un message s'affichera pour lui indiquer de remplir ce champ en particulier (cf. image 14).

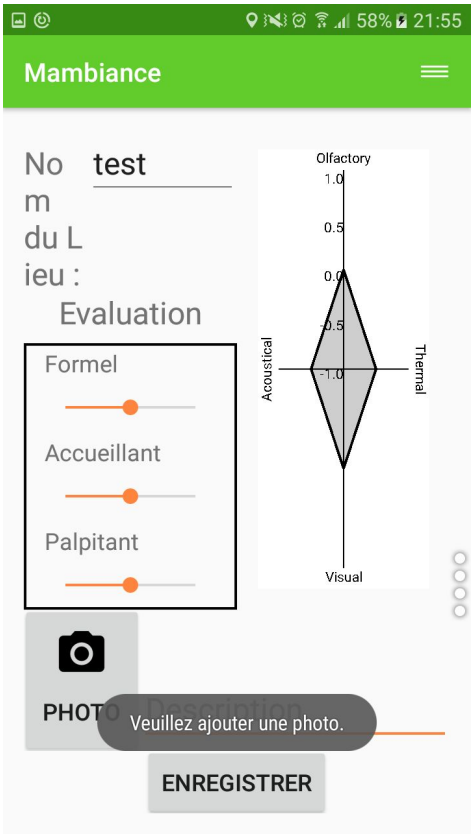


Image 14 : Message d'erreur pour l'importation d'une ambiance sans photo

Localisation de l'utilisateur

Auparavant, la localisation d'un utilisateur fonctionnait de façon relativement aléatoire, et si l'utilisateur cherchait à se localiser sans avoir activé son GPS ou sa connexion internet, l'application crashait.
Désormais, l'application gère ces exceptions, et des messages d'erreurs s'affichent pour demander à l'utilisateur d'allumer sa connexion internet (cf. image 15), ou d'allumer son GPS (cf. image 16).

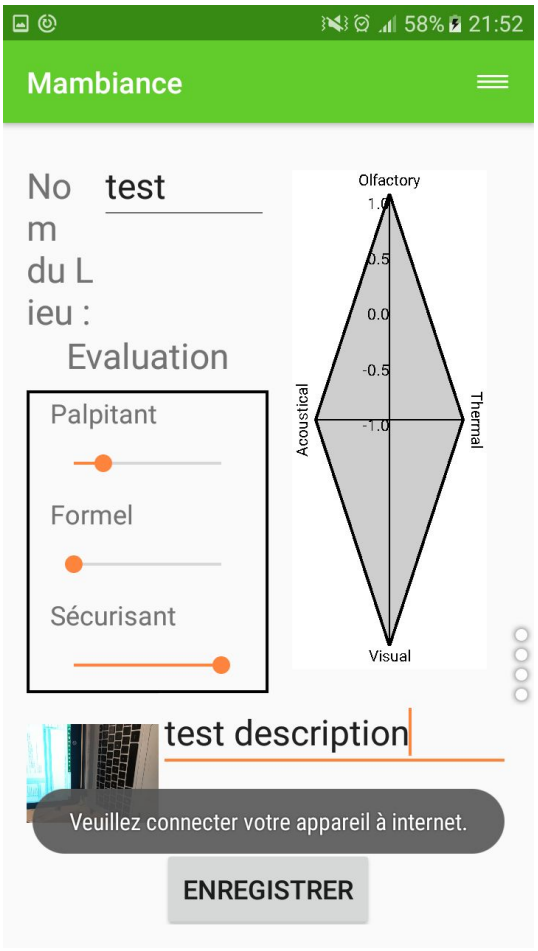


Image 15 : Message d'erreur lorsqu'il n'y a aucune connexion internet détectée

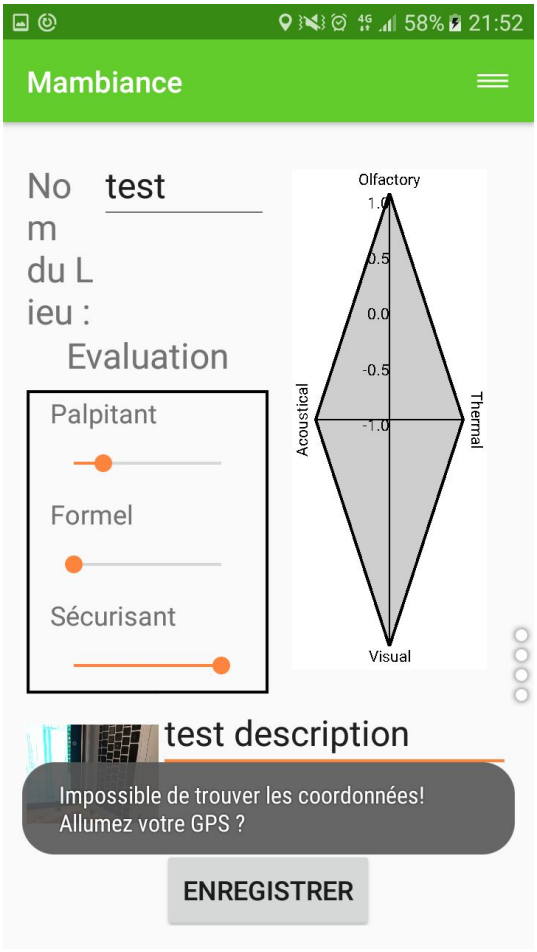


Image 16 : Message d'erreur lorsque les coordonnées sont introuvables

Ensuite, une pop-up s'affiche et l'utilisateur peut choisir de valider l'adresse proposée, ou d'entrer lui-même une adresse manuellement (cf. image 17). Si l'utilisateur remplit tous les champs obligatoires, l'application prend automatiquement en compte l'adresse qu'il a entrée, et non celle qui a été calculée à partir de sa position. Sinon, un message d'erreur s'affiche lui demandant de remplir toutes les cases.
Ceci est une nouveauté par rapport à l'ancien projet.

Un bouton recalculer permet de fermer la popup et de revenir en arrière.

Actuellement, des messages s'affichent une fois que l'utilisateur a choisi de valider l'adresse. Il s'agit de messages liés à l'envoi du marqueur au serveur distant, qui permettent de savoir si l'envoi s'est effectué sans encombres ou non.

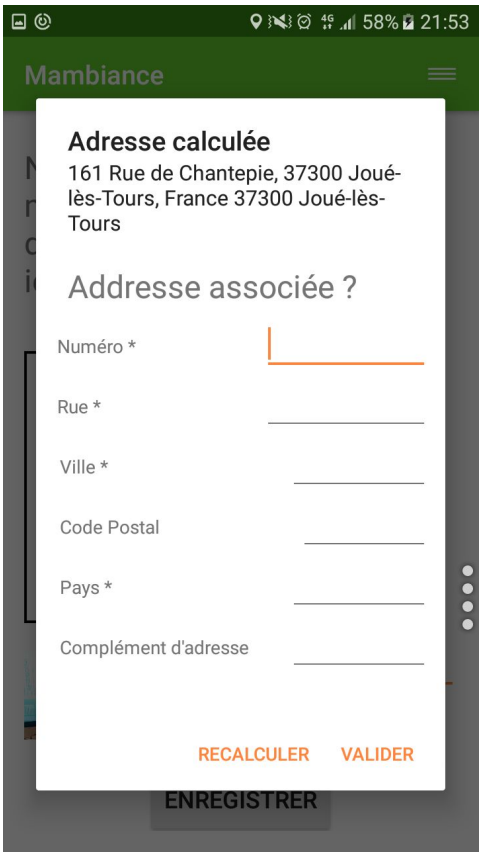


Image 17 : Popup pour valider l'adresse calculée ou en entrer une autre

Points d'amélioration

L'application ne vérifie pas qu'un lieu existe déjà. Elle créera un nouvel élément sans vérifier. Il faudrait également permettre de vérifier s'il existe une adresse déjà enregistrée pour le lieu localisé à laquelle on peut l'assimiler. Par exemple, pour une même place dans une ville, plusieurs localisations sont possibles (i.e. plusieurs coordonnées GPS). L'application devrait pouvoir permettre d'assimiler ces différentes coordonnées et donc ces différents lieux pris à une seule adresse.

Par ailleurs, l'application devrait, lors de la saisie d'une ambiance, récupérer les mots dont le booléen vaut 1 via une requête à la base de données distante. Ce n'est pas le cas actuellement, car l'application récupère aléatoirement des mots parmi ceux de la base de données locale dont le booléen vaut 1. Il y a donc un ajustement à faire ici.

De plus, les id de création ne sont pas ceux du serveur distant, mais ceux de l'appareil. C'est-à-dire que l'appareil va générer un nouvel id par rapport à ceux qu'il possède dans sa base locale, alors qu'il peut y avoir d'autres marqueurs en plus sur le serveur distant. Il faudrait donc récupérer l'id du prochain marqueur depuis le serveur distant et non depuis la base locale.

Par ailleurs, l'envoi de la photo entrée au serveur n'est pas gérée. Le processus est en effet complexe, et nous n'avons pas pu nous pencher dessus par manque de temps.

Enfin, à la première utilisation de l'application, celle-ci crashe puis demande l'autorisation à l'utilisateur d'autoriser l'utilisation de son GPS (cf. image 18). Une fois ceci accepté, le bug ne se reproduit plus.

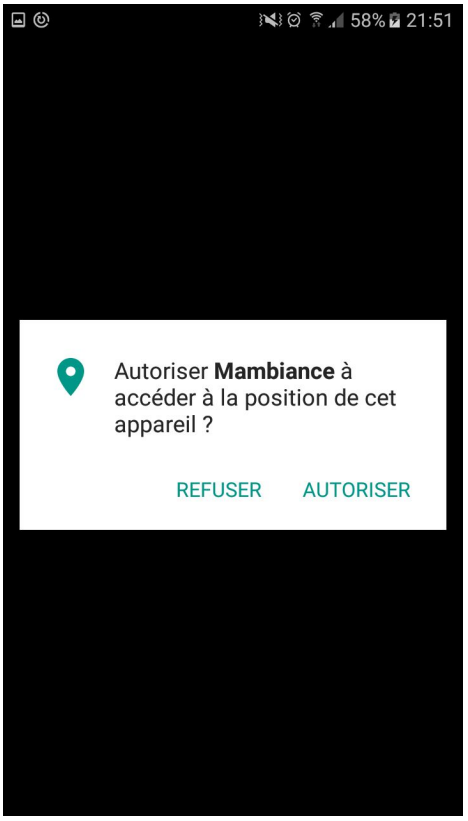


Image 18 : Message demandant l'autorisation d'accéder à la position de l'appareil

c. Carte des marqueurs

Dans le précédent projet, une carte permettait de visualiser les marqueurs associés à l'appareil utilisé.
Désormais, il y a deux cartes pour visualiser les marqueurs, soit deux activités différentes *MarkerActivity* et *MarkerActivityAll* qui présentent respectivement la carte des marqueurs entrés dans l'appareil utilisé, et la carte de tous les marqueurs du serveur (cf. image 19).

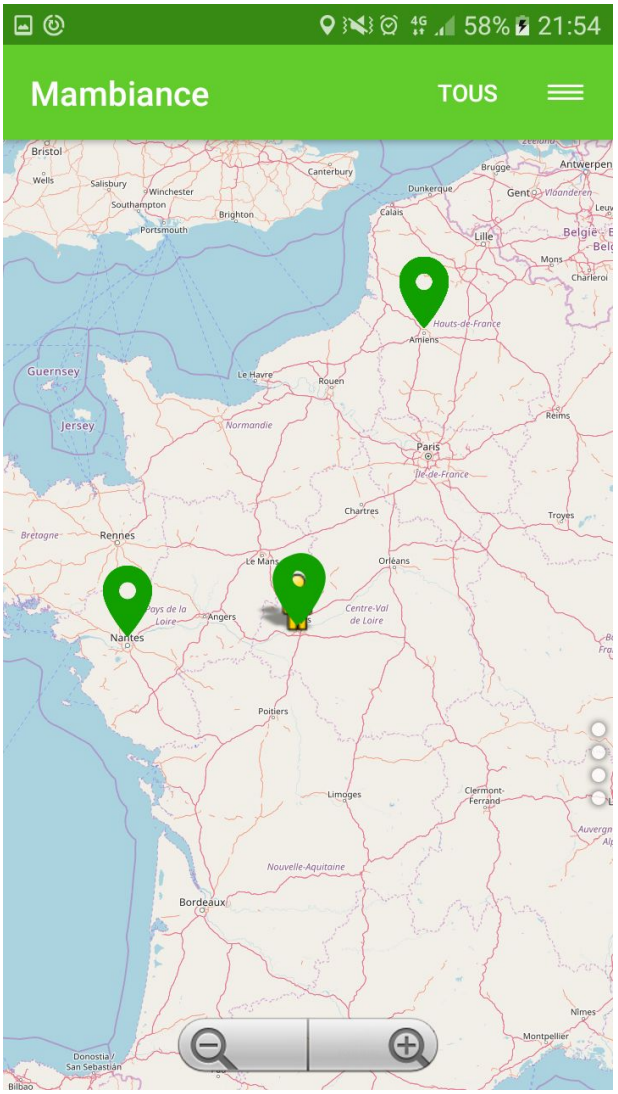


Image 19 : Carte de tous les marqueurs du serveur distant

Points d'amélioration

Depuis la carte de tous les marqueurs du serveur, il est possible de visualiser les informations liées aux marqueurs, mais seulement le nom, la description et la rose des ambiances associée au marqueur sélectionné (cf. image 20). Il faudrait donc effectuer un deuxième requête au serveur pour récupérer les mots et leurs notes associées.

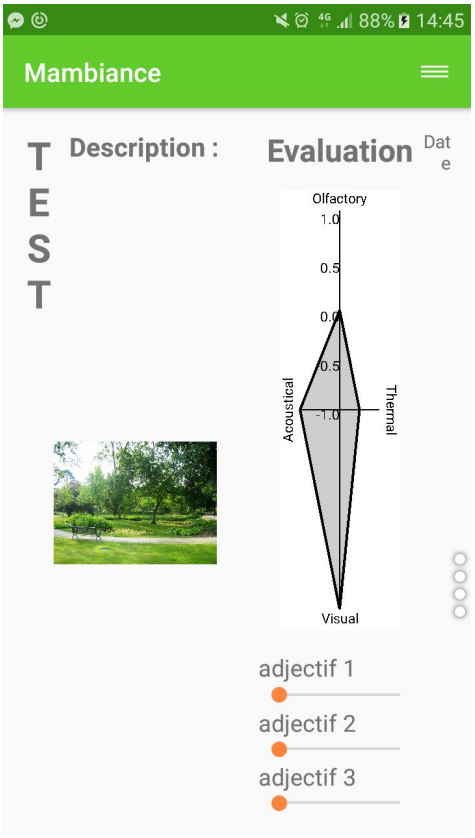


Image 20 : Ecran d'information d'un marqueur du serveur via la carte

d. Historique

L'historique n'a pas été modifié. Il affiche actuellement l'historique de l'appareil, c'est-à-dire les marqueurs qui ont été entrés via l'appareil utilisé, et non pas ceux associés à l'utilisateur. En effet, aucune connexion avec le serveur distant n'a été installée pour le moment.

9. Tests

a. Tests de l'API REST

Les requêtes MySQL ont été testées directement dans phpMyAdmin, puis ont été implémentées dans les fonctions PHP.

Les requêtes HTTP, une fois implémentées, ont été testées avec **Postman**.
Postman permet de créer des requêtes HTTP de manière graphique, en donnant le type de requête, l'adresse HTML, et les différents arguments.

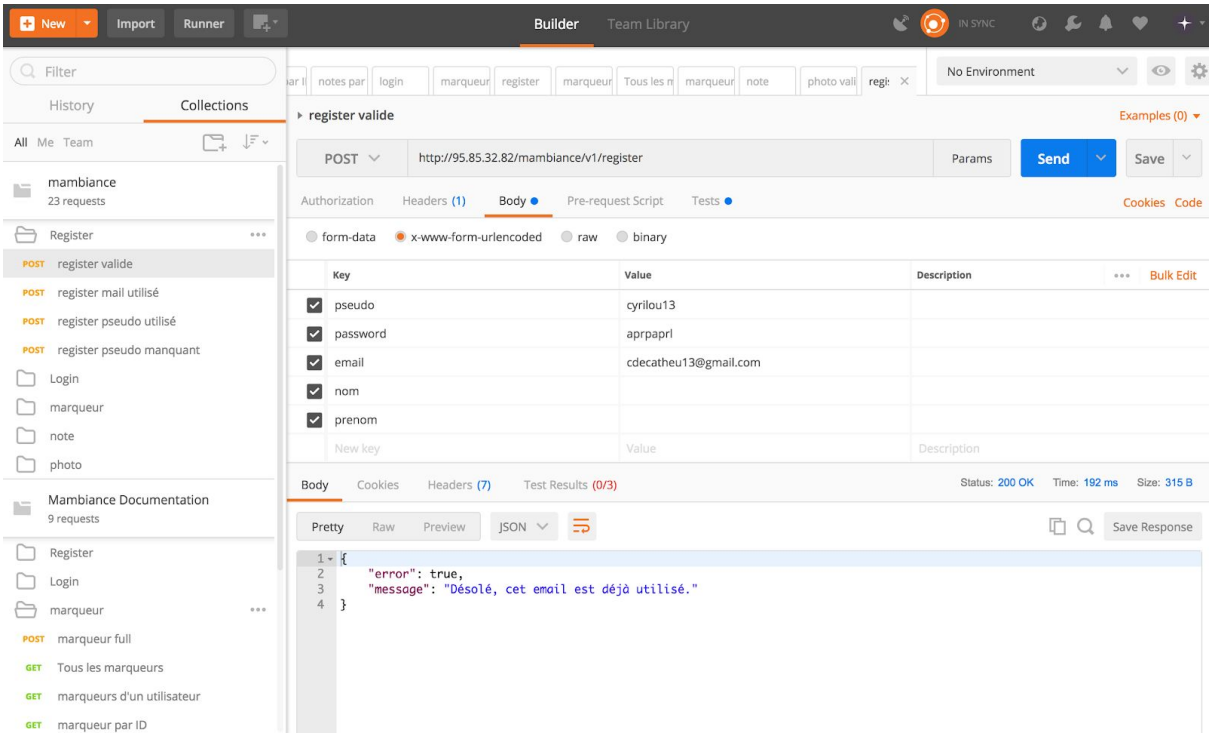


Image 21: Requête créée dans Postman. Résultat de la requête en bas à droite

Pour une fonctionnalité en particulier, plusieurs requêtes ont été réalisées pour vérifier tous les cas d'erreur.
Par exemple, la fonctionnalité Création d'un utilisateur possède 4 requêtes de test :

- une requête valide
- une requête valide mais avec un mail déjà utilisé dans la base de données
- une requête valide mais avec un pseudo déjà utilisé dans la base de données
- une requête valide mais avec un argument obligatoire non présent

Il est nécessaire que le comportement du middleware soit toujours connu et vérifié à chaque nouvelle évolution du code. Ceci permet de ne pas avoir de doute sur l'origine d'un bug : client ou middleware.
Les tests ont donc été automatisés : pour chaque requête, on analyse la réponse renvoyée par le middleware. Ces analyses ont été codées en **Javascript**, langage imposé par Postman.

Par exemple, pour la fonctionnalité Création d'un utilisateur, pour la requête valide, on vérifie 3 points :

- Le statut HTML de la réponse est 201 (signifie écriture dans la base de données)
- Le statut d'erreur est "False"
- Le message "Inscription réussie" apparaît bien dans la réponse

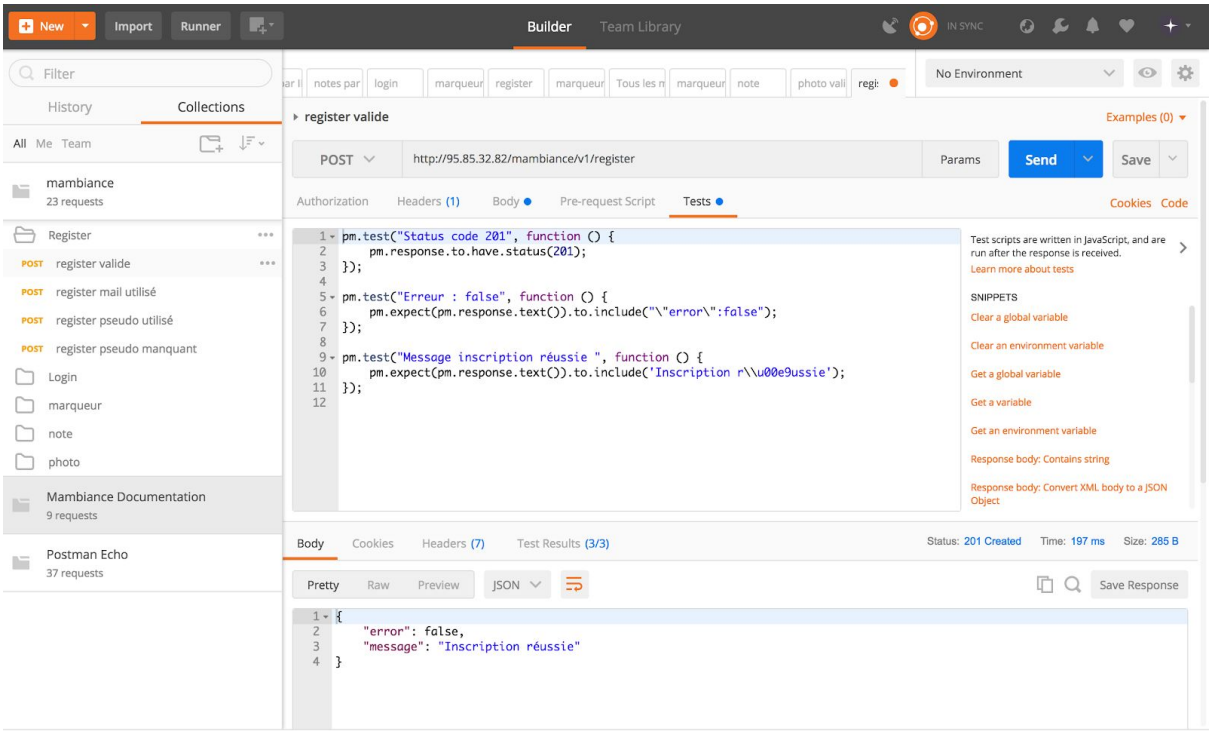


Image 22 : Partie de codage de l'analyse de la réponse d'une requête dans Postman

Nous avons ensuite lancé des jeu de tests régulièrement sur toutes les requêtes créées. Postman dans la fonction Runner réalise toutes les requêtes créées. Il renvoie ensuite les résultats des analyses Javascript.

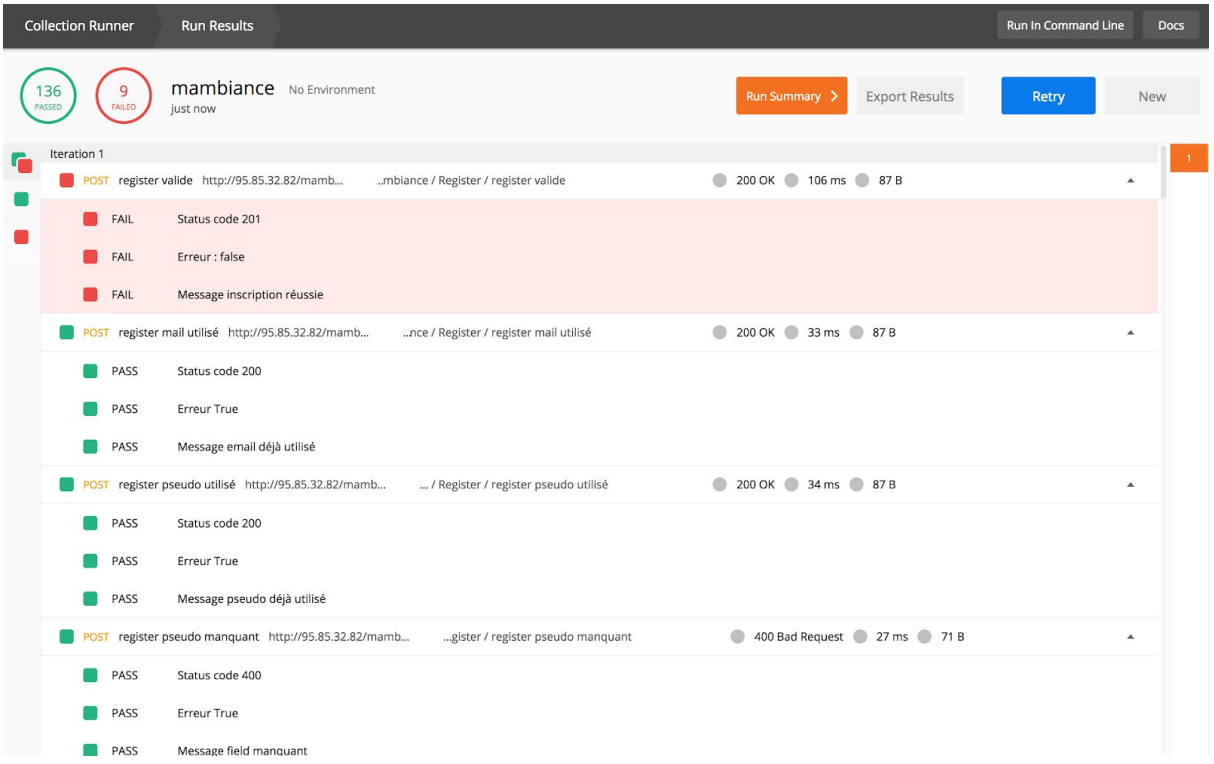


Image 23 : Jeu de tests. 23 requêtes, 136 analyses positives et 9 négatives

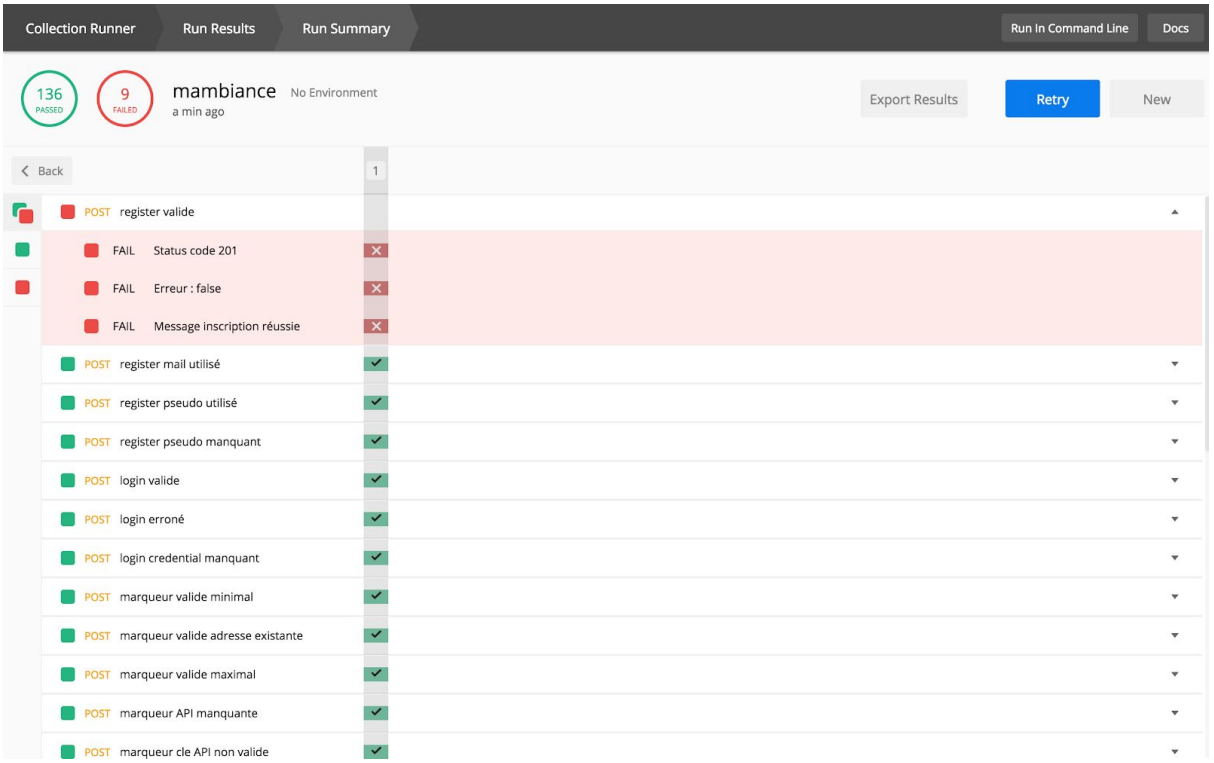


Image 24 : Version résumée du jeu de test

Le jeu de test et l'implémentation dans Postman est **le livrable principal** du middleware. C'est lui qui permet de générer la documentation vue dans la partie Fonctionnalités du Middleware et qui permet de générer du code pour Android.

Image 25 : Code généré automatiquement par Postman pour Android

b. Tests de l'application Android

Les tests ont été réalisés au fur et à mesure des modifications, directement sur un terminal Android, en utilisant le mode debug. Nous avons ainsi limité les risques de présence d'erreur.

La principale erreur restante est une exception non gérée par l'application : lorsque l'application se lance pour la première fois, pour les plus récentes versions d'Android, une demande d'autorisation d'accès à la position GPS est envoyée. L'application crashe alors, puis demande l'autorisation. Une fois l'autorisation accordée, il suffit de relancer Mambiance.

10. Livrables

En plus du rapport rédigé pour le projet, nous avons constitué un certain nombre de livrables:

- Notice d'utilisation pour des points de configuration importants (en Annexe)
- Documentation de l'API en production : https://documenter.getpostman.com/view/3191727/mambiance-documentation/7LkfiBj
- Export du jeu de requêtes de tests Postman : *mambiance-tests.postman_collection.json*
- Export du jeu de requêtes de documentation Postman : *Mambiance-Documentation.postman_collection.json*
- Export de la base de données : *mambiance.sql*
- Identifiants de la machine virtuelle : *droplet-credentials.txt*
- Identifiants de la base de données : *database-credentials.txt*
- Code source php : *mambiance-PHP-sources.zip*
- Code source Android : https://github.com/ppalmas/MAmbiance
- Fichier d'installation APK Android : *Mambiance.apk*

III. Gestion du projet

1. Planning

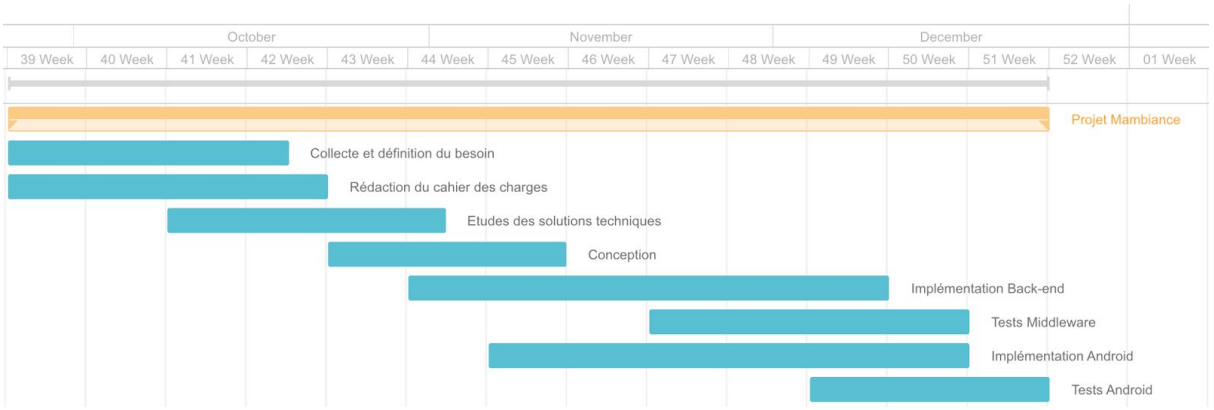


Image 26 : Gantt réel des tâches réalisées

Contrairement au Gantt prévisionnel, ce planning sépare les tâches back-end et front-end. Logiquement, le back-end a commencé plus tôt pour laisser le plus de temps possible au front-end.
Les tests middleware ont duré 2 semaines de plus qu'initialement prévu, car la charge de travail sur Postman avait été sous-estimée.

Les 4 premières semaines d'implémentation Android correspondent à du debuggage, à l'implémentation du nouveau modèle de données et à la mise en place des nouvelles fonctionnalités. Les tests étaient fait simultanément et sont compris dans ces semaines.
L'implémentation avec le middleware a commencé à la semaine 49. Les tests Android représentés dans le Gantt ne sont que les tests d'intégration avec le middleware.

Nous avions prévu une semaine de marge dans le planning, elle a été utilisée pour réaliser les derniers tests de l'application Android, rédiger ce rapport et mettre au propre les livrables.

2. Difficultés rencontrées

Nous avons rencontré diverses difficultés durant ce projet.

Il a d'abord été difficile de reprendre un projet existant. La première étape a été de prendre connaissance des rapports écrits des précédents projets. Conjointement, nous avons pris connaissance de l'interface Android et de son fonctionnement. Cette étape a été longue, mais grandement facilitée par une assez bonne documentation du code.
Nous avons pu nous rendre compte de l'importance de documenter clairement le code, mais aussi de la difficulté à rendre clair et intelligible cette documentation.

Par la suite, nous avons dû nous informer sur la connexion entre l'application et une base de données distante. Cette tâche a pris beaucoup de temps.
Nous avons également travaillé avec de multiples technologies, outils et plateformes, ce qui nous a pris un certain temps pour en comprendre le fonctionnement.

Nous avons passé également beaucoup de temps à déboguer puisque nous n'étions familiers avec aucune des technologies utilisées, et c'est quand le projet se termine qu'on devient apte à résoudre plus rapidement les bugs.

Par ailleurs, la création des tests pour le middleware a pris beaucoup plus de temps que prévu. La création de tests pour tous les cas possibles d'une requête, puis la mise en place de vérification automatique des résultats de requêtes prend plus de temps que la création de la requête elle-même !

Finalement, la principale difficulté à laquelle nous avons dû faire face a été de se former sur de multiples technologies à la fois et d'appliquer ces connaissances, en un temps très court.

IV. Bilan global

1. Améliorations possibles et perspectives

a. Interface

L'interface avait été faite dans l'ancien projet pour des tablettes plutôt que des smartphones android. Toutefois, beaucoup de fichiers XML de layouts sont définis en orientation verticale. Comme nous ne disposons pas de tablette, nous avons donc poursuivi un développement pour une orientation verticale plutôt que horizontale pour tablette.
Nous n'avons cependant pas modifié les layout du précédent projet. De fait, l'écran *Saisie d'un marqueur* n'est pas adapté à une orientation verticale type smartphone.

b. Saisie d'une ambiance et visualisation

Tout d'abord la saisie des ambiances peut être améliorée selon certains critères évoqués précédemment.

Par ailleurs, les critères de saisie d'ambiances pourraient évoluer. On pourrait par exemple afficher plus de 3 mots que l'utilisateur doit note, ne pas rendre la prise de photo obligatoire, ajouter un nouveau critère, envisager de nouvelles manières de "noter" un mot, etc.

Enfin, on pourrait ajouter des critères de visualisation des ambiances. L'utilisateur pourrait en effet vouloir visualiser sur une carte tous les marqueurs qui ont une note dépassant un certain seuil pour le mot 'Calme' par exemple, ou tous les marqueurs ajoutés durant l'hiver, etc.

c. Gestion des utilisateurs

Tout d'abord, la gestion des utilisateurs pourrait être améliorée comme décrit en II.2.a. Points d'amélioration. Les utilisateurs ne peuvent actuellement ni se connecter ni se déconnecter de l'appareil une fois inscrits ce qui pose un problème. De plus, ils ne peuvent changer leur mot de passe.

d. Création de marqueurs hors-ligne

A terme, il pourrait être intéressant de proposer aux utilisateurs une création de marqueurs hors-ligne, s'ils ne disposent pas dans l'immédiat d'une connexion internet. Il faudrait alors gérer l'envoi des informations vers le serveur distant manuellement (via une option d'envoi que l'utilisateur active par exemple).

e. Historique et carte

De la même manière qu'il existe une carte pour afficher tous les marqueurs du serveur, il pourrait être intéressant d'avoir deux historiques, l'un pour l'historique de l'appareil ou de l'utilisateur, et un autre pour l'historique de tous les marqueurs du serveur.

f. Nettoyage du code Android

Un travail de nettoyage et de clarification du code serait nécessaire.
Par exemple, le fichier LocalDataBase pourrait être séparé en plusieurs classes java - une par classe existante pour chaque table.
Par ailleurs, il y a de nombreuses répétitions de codes (notamment dans la classe *EditActivity.java*) qui pourraient être évitées en écrivant des méthodes dans des fichiers utilitaires.
Enfin, chaque string défini dans l'application pourrait être défini dans le fichier strings.xml. Cela permettrait de créer par la suite une version en anglais de l'application. Cela a d'ailleurs déjà été fait pour les fichiers *popup_address.xml*, *user_profile_activity.xml*, *user_activity.xml* et un *info_activity.xml*.

g. Serveur distant et base locale

Pour l'instant, les marqueurs et utilisateurs créés et envoyés à la base de données sont d'abord créés en local avant d'être envoyés au serveur. Du fait de cette pratique, les id des objets créés sont ceux de la base de données locale, et non du serveur distant. Si par exemple le prochain id à créer en local est 3, le prochain id à créer sur le serveur peut tout à fait être 15. Et alors l'objet créé aura l'id 3, ce qui pourrait poser problème.
De plus, cette pratique induit que, si l'enregistrement du marqueur sur le serveur rencontre un problème, le marqueur sera quand même enregistré en local. Il en va de même pour l'inscription d'un utilisateur.

h. Requêtes de modification et de suppression

Les requêtes de modification et de suppression de marqueurs n'ont pas été implémentées. La charge de travail est relativement faible pour un gain fonctionnel important. Néanmoins, il ne faut pas implémenter ces requêtes tant que le problème précédent n'a pas été résolu. Il y aurait trop de risque de suppression du mauvais marqueur. Ces requêtes nécessitent aussi la gestion des utilisateurs coté Android pour que les utilisateurs ne puissent écrire que sur des marqueurs qu'ils possèdent.

i. Requêtes de réinitialisation de mot de passe

La réinitialisation d'un mot de passe implique l'envoi d'un mail contenant un jeton de réinitialisation. La création d'un jeton n'est pas très compliquée, mais la mise en place d'un serveur de mail l'est beaucoup plus : dépendance à un serveur de mail existant, intégration d'un champ variable dans le mail, exigence de performance (le délai d'un mail de réinitialisation doit être de moins de 3 minutes).

j. Consolidation de la documentation de l'API

La documentation de l'API n'a en exemple de réponses que des réponses positives. Les exemples de réponses négatives ne peuvent pour l'instant être trouvés que dans le jeu de test total de l'API. On pourrait compléter la documentation en mettant aussi des exemples de réponses négatives quand on voudra perfectionner la gestion des erreurs par le client.

k. Recommandations générales

Enfin un problème récurrent avec Android est de gérer les versions. Lorsque nous avons repris le projet, les mises à jour d'Android empêchaient l'accès à l'appareil photo ainsi que le stockage des photos, et la géolocalisation ne fonctionnait plus. Il faut donc faire particulièrement attention aux conséquences des mises à jour d'Android, et savoir avec quelles versions d'Android l'application est compatible.

2. Apports personnels

En ce qui concerne le travail d'équipe, nous n'avons jamais eu à travailler sur les mêmes codes. Ceci est dû à l'organisation classique pour ce type de projet qui consiste à séparer les développeurs back-end et les développeurs côté client. Nous avons bien veillé à nous mettre d'accord sur les périmètres fonctionnels de nos parties avant de se lancer dans les développements.

a. Paola

Ce projet a été l'occasion pour moi de coder pour Android en reprenant un projet déjà existant, ce qui était une première. C'était complexe mais très intéressant et enrichissant de reprendre un projet, de le comprendre et de s'y adapter. Le debugger très efficace d'Android Studio m'a permis d'apprendre pas à pas.

J'ai appris ainsi de nouvelles techniques de codage, l'architecture d'un projet Android, et l'utilisation d'une base de données locale SQLite. J'ai également découvert la gestion de projet qui consiste à séparer le développement en back end et front end.

J'ai découvert finalement le fonctionnement général pour faire communiquer l'application Android et une base de données distante notamment puisque j'ai dû gérer cette connexion au niveau de l'application. De ce fait, j'ai notamment appris à gérer les requêtes HTTP depuis Android, et à traiter le format JSON.

b. Cyril

Je me suis chargé de la partie back-end : serveur, base de données et middleware.
Pour la partie serveur, j'ai appris les bases du métier de Sysadmin : configuration d'un serveur, gestion des versions et des connexions. La mise en place d'une base de données m'a permis de mettre en pratique, et avec un nouvel outil, phpMyAdmin, l'administration d'une base de données.

C’est le travail sur le middleware qui m’a le plus apporté : j’ai dû choisir la solution technique la plus pertinente au vu de mes connaissances limitées dans le domaine, puis prendre en main un nouveau langage, PHP, et un nouveau framework. J’ai dû avoir une approche orientée tests pour fournir à Paola des requêtes documentées avec des comportements totalement connus même en cas d’exception ou d’erreur et prendre en main Postman, outil très puissant mais très dense.
Sur l’ensemble du projet, j’ai aussi dû apprendre à faire des choix de conception : par exemple, où mettre une contrainte d’intégrité : au niveau de la base de données, du middleware ou du client. Il faut alors réfléchir à tous les impacts que cela peut avoir.

V. Annexes

1. Cahier des charges

Cahier des charges

Projet
Mambiance

Equipe: Cyril de Catheu, Paola Palmas

Encadrants: Myriam Servières, Vincent Tourre

Table des matières

I. Présentation du projet

1. Contexte

Ce projet s'inscrit dans le cadre du module PAPPL, projet d'application 1, dispensé en option Informatique. Il dure environ trois mois entre le 15 septembre et le 22 décembre 2017.
Ce projet fait suite à deux projets Mambiance datant de 2014 et 2016. Le premier projet a permis la conception d'une application android permettant de caractériser l'ambiance d'un espace urbain au moyen de marqueurs (mots, images, curseurs). L'idée était de permettre à un utilisateur de qualifier l'ambiance qu'il ressentait à un instant donné dans un espace urbain donné, et de pouvoir partager ce ressenti par le biais d'un réseau social. Le second projet a permis d'y ajouter un nouveau marqueur, la rose des ambiances. La première équipe a rendu une application construite mais non opérationnelle. La seconde équipe a d'abord modifié le modèle de données de la base de données, et rendu opérationnelle l'application.

2. Définition du projet

Le projet Mambiance repose ainsi sur l'association de lieux à des ambiances, décrites par les ressentis et sensations des utilisateurs à l'aide marqueurs proposés par l'application.
La base de données accompagnant l'application devra être déplacée vers un serveur externe pour permettre aux utilisateurs de pouvoir partager ces ambiances. La base de données est actuellement en local sur le téléphone ou la tablette utilisant l'application.

Cette année, le projet consiste en particulier à la mise en place d'un serveur et d'une base de données distantes pour l'application. Il sera surement nécessaire de proposer des améliorations de l'application, modifier la conception de la base de données, et concevoir la gestion de comptes utilisateurs.

3. Diagnostic

A ce jour, l'application est fonctionnelle. Cependant, plusieurs bugs empêchent la bonne utilisation de l'application. De plus, la gestion des différents utilisateurs n'est pas implémentée, puisque la base de données se trouve actuellement sur un serveur local.

Nous avons effectué des tests d'abord sous émulateur sur ordinateur, puis directement sur un téléphone sous android.

Base de données

La base de données se trouve actuellement sur un serveur interne à l'appareil utilisé. Cette configuration ne permet pas une gestion de plusieurs utilisateurs qui posteraient plusieurs ambiances sur la base en vue de partager leurs ressentis.
Par ailleurs, la base de données, du fait de sa configuration, ne gère pas plusieurs marqueurs associés à un même lieu. Si deux ambiances ont été saisies pour un même lieu, mais que leurs coordonnées sont légèrement différentes, l'application ne les proposera pas comme des ambiances d'un même lieu.

Fonctionnalités

L'utilisateur ne rencontre aucun problème lors de l'ouverture de l'application et l'interface est bonne.
En revanche, lors de la saisie d'une ambiance dans le formulaire, de nombreux bugs apparaissent. Dans une interface verticale, l'écran est illisible et manque d'ergonomie.

Fonctionnalités pour la saisie d'une ambiance	**Bugs**	**Interface**
Saisie d'un mot	Oui	Mauvaise, le clavier cache l'écran
Saisie de la rose des ambiances	Non	Bonne
Prise d'une photo	Oui	L'application bug et s'arrête lorsque l'utilisateur veut prendre une photo
Ajout de l'ambiance	Oui	L'application s'arrête lors de la recherche des coordonnées géographiques
Historique des ambiances ajoutées	*Non testé*	*Non testé*
Interface	Oui	Mauvaise ergonomie, Clavier impossible à quitter après ouverture

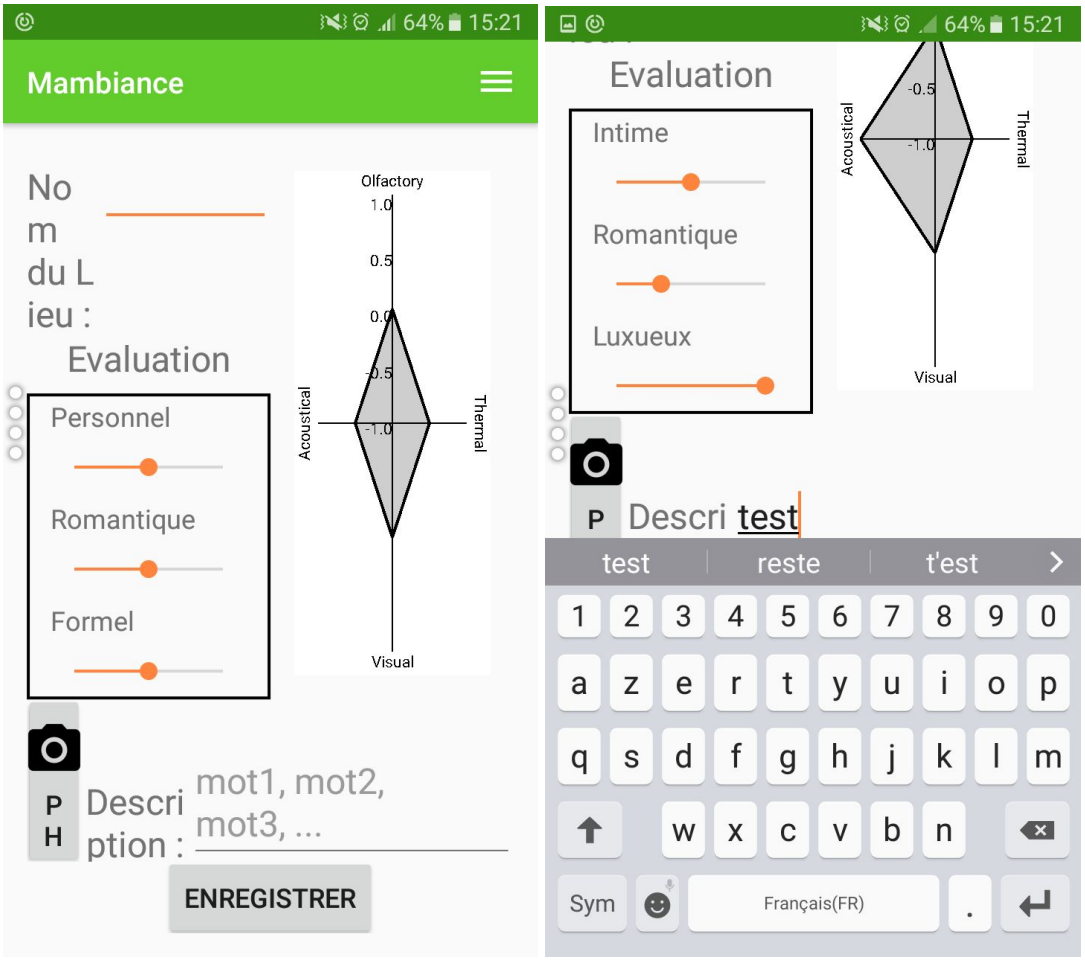


Figure 1. Captures d'écran de l'application Mambiance

II. Besoins et objectifs

1. Prestations attendues

L'objectif de cette partie est de présenter les différentes fonctionnalités attendues pour cette application. Elles sont au nombre de 5.

Fonction 1 : Profil utilisateur

Objectif : L'utilisateur doit pouvoir créer, modifier et supprimer son profil.

Description :

Le profil devra contenir les informations suivantes:

- Nom
- Prénom
- Adresse mail
- Mot de passe

Le système doit vérifier que l'adresse mail est valide et n'est pas déjà utilisée. Celle-ci constituera l'identifiant de l'utilisateur.

Fonction 2 : Saisie de marqueurs

Objectif : L'utilisateur doit pouvoir saisir sur l'application des données qualifiant l'ambiance d'un lieu.
Description :
L'utilisateur saisit les données suivantes sur l'interface :

- Une photo
- Un mot
- Trois critères à évaluer sur une échelle
- Une appréciation via la rose des ambiances

Si l'utilisateur n'a pas rempli un des champs ci-dessus, l'application affiche un message d'erreur qui indique ce qu'il manque. Chaque critère est obligatoire (à valider).
Les informations suivantes doivent être récupérées par l'application :

- Une position géographique
- Une information temporelle (date et heure)

Toutes ces informations sont enregistrées dans une base de données.

Fonction 3 : Affichage d'une carte des marqueurs
Objectif : L'utilisateur doit pouvoir accéder à une carte montrant à l'aide de marqueurs les différents lieux auxquels des ambiances ont été saisies.
Description :
La carte n'affiche qu'un marqueur par lieu i.e. par adresse. L'utilisateur sait alors quels lieux ont déjà été qualifiés par d'autres personnes.

Fonction 4 : Affichage des informations liées aux marqueurs
Objectif : L'utilisateur doit pouvoir accéder aux informations liés aux différents marqueurs visibles sur la carte
Description :
Lors d'un clic sur un marqueur, l'utilisateur a accès à la liste des marqueurs enregistrés à cet endroit, représentés par un mot (le nom du lieu), une date et un horaire. Ensuite, en cliquant sur un marqueur, il accède aux différents critères liés à celui-ci.

Fonction 5 : Edition des marqueurs
Objectif : L'utilisateur doit pouvoir modifier ou supprimer un marqueur qu'il a lui-même précédemment saisi.
Description :
Il faut permettre à l'utilisateur d'accéder à une liste des marqueurs qu'il a saisi, afin qu'il puisse modifier les différents critères qu'il a saisis, à savoir:

- La photo
- Le mot
- L'évaluation des trois critères
- L'évaluation via la rose des ambiances

2. Contraintes

2.1 Contraintes techniques

Contrainte 1 : Développement Android
La technologie utilisée pour développer l'application Mambiance est la technologie Android. Puisqu'il s'agit du système d'exploitation mobile le plus utilisé au monde, l'application pourra être utilisée sur un grand nombre de tablettes ou smartphones. Son utilisation est entièrement conçue pour ce genre d'appareils.

Contrainte 2 : Base de données spatiale et distante
Mettre la base de données sur Serveur externe.
MySQL ou PostgreSQL, Base de données spatiale

Contrainte 3 : Open Street Map
Le projet précédent a utilisé Open Street Map à la place de Google Maps car il s'agit d'une base de données géographique libre. Nous continuerons donc d'utiliser Open Street Map. Si les API d'Open Street Map ne proposent pas une fonctionnalité pour zoner un ensemble de coordonnées comme un lieu unique, nous étudierons la possibilité de revenir à Google Maps.

2.2 Contraintes légales

Les données personnelles demandées par l'application sont : un nom, prénom, une adresse email, et la localisation de l'appareil utilisé lorsqu'une ambiance est saisie. Ces données seront stockées sur une base de données distante appartenant à l'Ecole Centrale de Nantes. Nous nous assurerons que la base de données soit conforme au exigences de la Loi Informatique et Liberté et du Règlement Général sur la Protection des Données qui entrera en vigueur. Nous afficherons aussi un texte expliquant que les données entrées par l'utilisateur ne sont pas transmises à d'autres services, et que la position de l'utilisateur n'est enregistrée que lorsqu'il crée un marqueur.

3. Objectifs

Nous avons listé succinctement quelques objectifs à atteindre pour ce projet. Cependant, certains ne seront peut-être pas atteignables par manque de temps, en particulier les derniers.

Ordre de priorité	**Objectif**	**Description**
1	Refonte de la base de données	Modification du modèle de données, notamment pour ajouter l'utilisateur, position géographique permettant de rassembler plusieurs marqueurs.
2	Création de nouveaux marqueurs	Choix de nouveaux marqueurs ou modification des précédents, permettant de qualifier une ambiance urbaine
3	Création d'une base de données distante	Création d'une base de données distante sur le serveur de l'école, basée sur le modèle de données établi
4	Corrections des bugs	Correction des bugs liés à la prise de photo et à la géolocalisation par l'application
5	Création d'une interface pour la création du profil utilisateur	Implémentation d'une interface pour la création du profil utilisateur à partir du menu
6	Modification de l'interface	Modification de l'interface pour l'écran de saisie des marqueurs

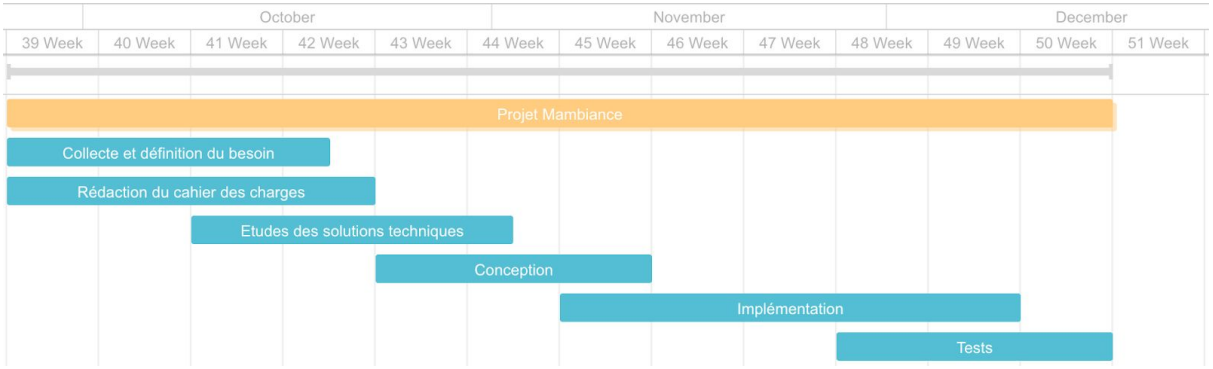
III. Déroulement du projet

1. Choix de la méthode de gestion de projet

Nous fonctionnerons sur un modèle semi-itératif en V : nous décomposerons les sujets en sous-sujets, sur lesquels nous appliquerons la méthode en V. Nous essaierons de traiter le plus possible de sous-sujet en le temps imparti. Les sous-sujets seront priorisés par les encadrants en fonction de leur besoins ou par l'équipe en fonction de ses préférences.

2. Planning prévisionnel

Le planning prévisionnel présenté ci-dessous est sujet à évolution dans les 3 semaines à venir. Nous avons gardé une semaine de marge pour le moment en cas d'impondérable.



Gantt prévisionnel

2. Notice d'utilisation

a. Création et disponibilité de mots

L'application permet de donner des notes à des mots. Il n'a pas été développé d'interface graphique administrateur pour gérer les mots. Pour ajouter un mot, il faut le créer dans l'interface d'administration phpMyAdmin.

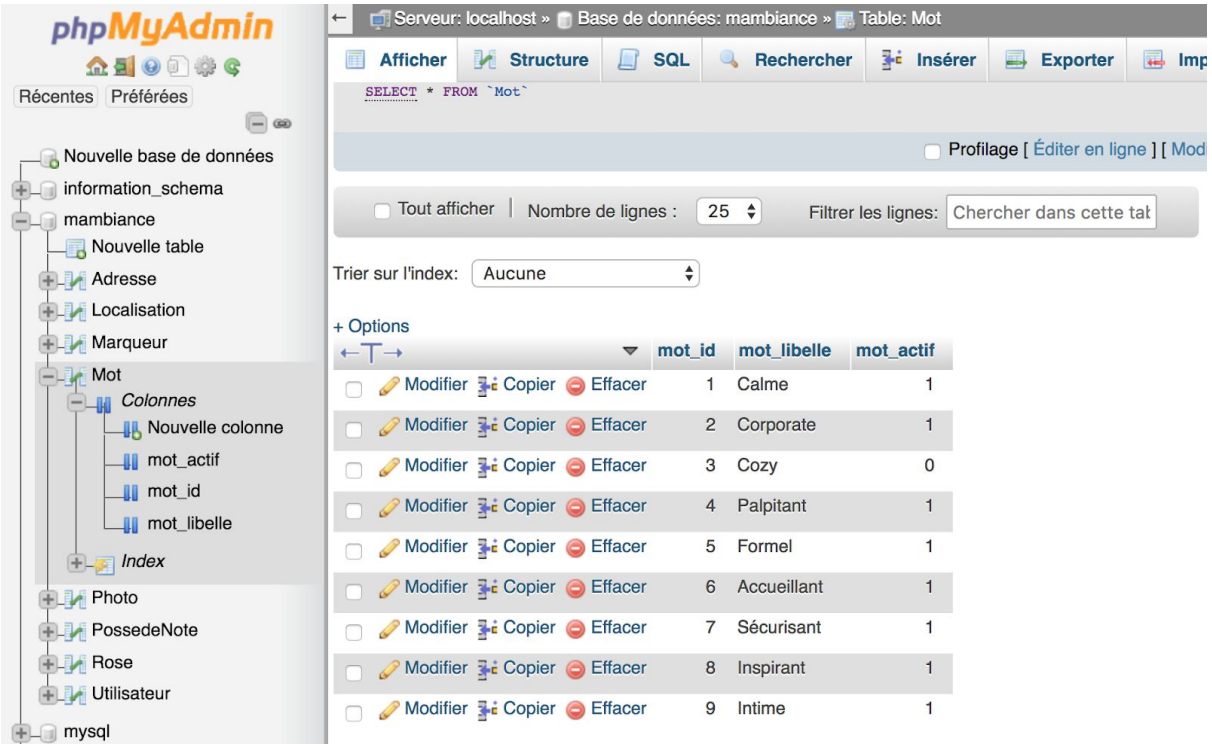


Image 27 : Liste des mots dans l'interface d'administration phpMyAdmin

Donner la valeur suivante pour mot_actif :

- 1 : le mot peut apparaître dans l'application Android. (valeur par défaut)
- 0 (ou autre valeur) : le mot ne peut pas apparaître dans l'application Android

b. Configuration du middleware pour une autre base de données

Seul le fichier config.php contient des informations relatives à la base de données utilisée. C'est ce fichier qu'il faudra modifier lors de la mise en place sur un nouveau serveur avec une nouvelle base de données.

```php
<?php
/**
 * Database configuration
 */

define('DB_USERNAME', 'root');
define('DB_PASSWORD', 'a6a75850b8b44d549476e85a866185ba4215979b861a1969');
define('DB_HOST', '95.85.32.82');
define('DB_NAME', 'mambiance');
```

Image 28 : Variable à changer dans *config.php*

Ne pas changer le nom des variables, mais seulement leur valeur:

- DB_USERNAME : nom de l'utilisateur pour la base de données
- DB_PASSWORD : mot de passe de la base de données
- DB_HOST : adresse de la base de données
- DB_NAME : nom de la base de données

c. Configuration du serveur apache

L'ensemble des configurations du serveur apache est dans le fichier *apache2.conf*, rendu dans les livrables.

Noter que 3 autres actions de configuration ont été réalisées :

1) Le module "Apache rewrite" a été activé. Cela permet une compatibilité avec le raccourcissement des URL par le framework Slim PHP.
Pour activer ce module sur un serveur apache 2, exécuter la commande :
sudo a2enmod rewrite

2) Pour se connecter à une base MySQL, le serveur Apache a besoin de drivers. Les drivers mysqlnd ont été installés.
Pour installer ces drivers sur un serveur apache2, exécuter les commandes :
sudo apt-get update
sudo apt-get install php5-mysqlnd

3) Le fichier *apache2.conf* situé dans /etc/apache2 a été modifié.
Mettre la valeur de "AllowOverride" à "All"

Ces 3 étapes sont décrites dans le tutoriel suivant :
https://www.androidhive.info/2015/03/android-hosting-php-mysql-restful-services-to-digitalocean/

d. Configuration PHP avec Apache

L'ensemble des configurations du php lié au serveur apache est dans le fichier *php.ini*, rendu dans les livrables.
Pour une installation classique, ce fichier est situé dans */etc/php/7.0/apache2*.
C'est dans ce fichier qu'est spécifié le chemin du dossier d'upload temporaire des images. Il est important que le chemin soit valide pour que l'upload de fichier fonctionne. Le dossier en question doit être accessible en lecture et en écriture par apache. Il sera peut-être nécessaire de régler les droits du dossier en question en fonction de son emplacement et des règles de sécurité du système utilisé.

e. Installation des fichiers php sur un serveur

Sur une configuration Linux LAMP 16.04 ou similaire, avec Apache2 installé :

- décompresser l'archive de source *mambiance-PHP-sources.zip*
- placer le contenu de l'archive sur votre système à l'adresse */var/www/html/*

f. Clé API dans l'application Android

La Clé API est entrée manuellement dans l'application Android. Lors de la création de l'utilisateur, une clé API est enregistrée sur la base locale. La clé entrée ci-dessous automatiquement correspond à un utilisateur enregistré actuellement sur le serveur distant. A chaque création d'un utilisateur sur la base distante, une clé lui ai attribuée.
La Clé est nécessaire, comme dit plus haut, pour des questions de sécurité. Lorsque la connexion de l'utilisateur à son compte sera programmée, il faudra faire attention à cette clé qu'il faudra alors récupérer pour assurer la connexion.

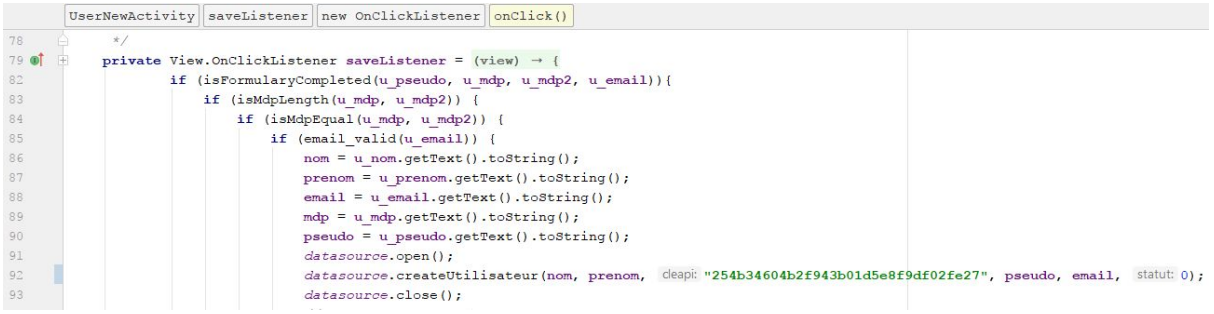


Image 29 : Impression d'écran de la création Utilisateur

g. Adaptation de l'application à un nouveau serveur

```
//Envoi au serveur distant
OkHttpClient client = new OkHttpClient();
//Création de la requête
MediaType mediaType = MediaType.parse("application/x-www-form-urlencoded");
RequestBody body = RequestBody.create(mediaType, content: "pseudo=" + pseudo + "&password=" + mdp + "&email=" +
        email + "&nom=" + nom + "&prenom=" + prenom);
Request request = new Request.Builder()
        .url("http://95.85.32.82/mambiance/v1/register")
        .post(body)
        .addHeader( name: "Content-Type",  value: "application/x-www-form-urlencoded")
        .addHeader( name: "Cache-Control",  value: "no-cache")
        .addHeader( name: "Postman-Token",  value: "3fa60859-6dd0-9e78-d6ac-d64aa9a65c99")
        .build();
```

Image 30 : Capture d'écran de l'envoi d'un Utilisateur à la base distante

Dans l'implémentation Android, chaque requête utilise un url. Lors du changement vers un nouveau serveur, il faudra modifier l'url (http://95.85.32.82/mambiance/v1/register dans l'exemple ci-dessus) à cet endroit, ainsi que la clé API. A l'avenir, une solution plus pratique est de créer une variable URL qui sera plus facilement modifiable.

3. Bibliographie, guides et tutoriels

Nous avons largement utilisé les précédents rapports de projet, et nous nous sommes basés sur la précédente application Mambiance.

Nous avons également consulté de la documentation en ligne et des tutoriels :

Section développeur d'Android, surtout pour sa documentation :
https://developer.android.com/index.html
Tutoriels en ligne sur Android :
https://kosalgeek.com/
Tutoriel en ligne sur le format JSON :
https://www.androidhive.info/2012/01/android-json-parsing-tutorial/
https://www.androidhive.info/2012/01/android-json-parsing-tutorial/
Tutoriel en ligne sur la connexion entre android, php et mysql, notamment pour l'utilisation des libraires :
https://zestedesavoir.com/tutoriels/1140/communication-entre-android-et-php-mysql/
http://tutorielandroid.francoiscolin.fr/bdd.php#archiexterne
Librairie OkHttp :
http://tutos-android-france.com/introduction-a-okhttp/
http://www.vogella.com/tutorials/JavaLibrary-OkHttp/article.html
Création d'une API REST avec MySQL, PHP et Slim PHP :
https://www.androidhive.info/2014/01/how-to-create-rest-api-for-android-app-using-php-slim-and-mysql-day-12-2/

https://www.androidhive.info/2014/01/how-to-create-rest-api-for-android-app-using-php-slim-and-mysql-day-23/
Bases, structures de données et opérateurs en PHP:
https://www.w3schools.com/php/php_syntax.asp
Hébergement distant et paramétrage serveur :
https://www.androidhive.info/2015/03/android-hosting-php-mysql-restful-services-to-digitalocean/
Accès à un serveur MySQL depuis n'importe quel adresse IP avec l'utilisateur root:
https://forum.ubuntu-fr.org/viewtopic.php?pid=2950231#p2950231
Mise en place et optimisation du fichier config.php:
https://stackoverflow.com/questions/1712973/how-to-include-config-php-efficiently
Type de formulaire d'envoi en requête POST pour uploader une image:
https://stackoverflow.com/questions/4249609/form-content-type-for-a-json-http-post