JUIN 2024

UE CEPI

RAPPORT PROJET

Maxence Péron
Gaspard Langlais
Gautier Vasse
Vanessa Chambon
Leelou Lebrun

Table des matières :

I. Expression du besoin

Lors d'un évènement, il n'est pas rare de rencontrer des gens avec qui l'on voudrait garder contact pour pouvoir reconnecter avec eux plus tard. Cependant il peut être difficile de retrouver ces personnes rencontrées brièvement sur les réseaux sociaux sans informations spécifiques.

EMAMI souhaite donc faciliter les connexions sociales en permettant à ses utilisateurs de retrouver les informations des personnes croisées la veille.
Le principe est simple :

- L'application affiche les profils de toutes les personnes avec qui vous avez passé du temps dans les 24h précédentes
- Vous avez ensuite la possibilité de les demander en ami
- S'il y a réciprocité, vous avez accès à ses différents profils de réseaux sociaux

II. Organisation globale du projet

1) Equipe

GASPARD LANGLAIS
TheGasp

GAUTIER VASSE
GauTit

VANESSA CHAMBON
vanessa430

LEELOU LEBRUN
LebLeelo

MAXENCE PÉRON
maxenceperon

2) Méthodologie

Les premières séances ont été dédiées à la définition de notre application, notamment concernant son fonctionnement général.
Cette partie du projet n'a pas été réactualisée durant la progression du projet, ce qui a induit des erreurs de lien entre les différentes parties en fin de projet.

Pour être mené à bien, le projet a ensuite été réparti en trois grandes parties :

- L'interface, sur laquelle ont travaillé Vanessa Chambon, Gaspard Langlais et Leelou Lebrun
- La base de données, sur laquelle a travaillé Maxence Péron
- La connexion Bluetooth sur laquelle ont travaillé Gaspard Langlais et Gautier Vasse

Les tâches à effectuer pour chaque personne ont ensuite été inscrites dans le Backlog de GitHub. Nous n'avons cependant pas affiné suffisamment les tâches et n'avons pas défini la longueur de nos sprints, ce qui a grandement ralenti le projet.

Le projet a été réalisé sur AndroidStudio et lié à GitHub.
Lien du projet : https://github.com/TheGasp/EMAMI.git

III. Description fonctionnelle

1) Fonctionnalités principales

Historique des rencontres

L'application enregistre les rencontres récentes pour que l'utilisateur puisse venir les consulter ultérieurement. Cet inventaire est affiché sur la page principale de l'application sous forme de profil réduit contenant simplement la photo de la personne et son nom et prénom. Il est conservé pendant 24h.

Demande d'amis

Si une personne apparait dans cet inventaire, il est possible de la demander en ami. Si à son tour, cette personne vous demande en ami, vous êtes alors en capacité de consulter son profil complet où sont référencés tous ces réseaux et éventuellement son numéro de téléphone.

2) Interface utilisateur

Pour un utilisateur déjà inscrit, l'application se présente de la manière suivante :

Ecran d'accueil
Cette page présente les différents profils des personnes que l'on a croisé. Elle permet de swiper à gauche ou à droite selon que l'on veuille demander la personne en ami ou non. Pour une plus grande efficacité, l'interface est simple et intuitive, composée d'icônes claires.

Paramètres
Il est possible pour l'utilisateur de changer à tout moment ses informations. En allant dans les paramètres, il peut choisir de modifier, supprimer ou ajouter des informations. Il peut également choisir quelles informations afficher ou cacher.

Pour un utilisateur encore non inscrit, ou qui veut se connecter :
Les écrans de connexions
L'utilisateur peut y rentrer toutes ses informations pour créer son compte s'il n'est pas inscrit, ou se connecter. Il est également possible de réinitialiser le mot de passe.

IV. Description de l'architecture du projet

1) Interface

a. Interface liée à la connexion – Vanessa Chambon

Interface de connexion

Cette interface est la première page à laquelle l'utilisateur accède en ouvrant l'application. Elle lui permet soit de se connecter à son compte, soit de créer un compte.

Il se connecte en complétant son adresse e-mail et son mot de passe puis appuyer sur le bouton de connexion.
Il existe deux autres boutons : « Mot de passe oublié ? » et « Créer un compte » qui redirige l'utilisateur vers une page suivante.
Elle est sous forme d'un fichier xml « connexion.xml »

Elle est composée :
- D'une Classe Java MainActivity qui donne l'action des boutons : ouvrir les pages associées,
- De deux LinearLayout superposés afin de pouvoir moduler les éléments sur la page comme on le souhaite.

Pour le mot de passe, celui-ci est masqué mais l'utilisateur à la possibilité de le visualiser grâce à cette commande : android:inputType=« textPassword »

Interfaces création de compte

Pour créer son compte, l'utilisateur devra remplir deux pages ; l'une avec ses informations personnelles puis l'autre avec ses identifiants sur les réseaux sociaux qu'il utilise.

La Classe Java « Creation » est associée à « creationcompte.xml » et la Classe Java « Reseau » est associée à « reseau.xml ». Ces deux pages sont reliées par un bouton « COUTINUER ».
Elles sont, elles aussi, construites avec des superpositions de LinearLayout et elles possèdent un bouton « BACK » leur permettant de revenir en arrière.

Lorsque l'utilisateur rentre son mot de passe, il ne possède pas la possibilité de le visionner car il doit le confirmer en dessous, cela renforce la sécurité.
Sur la page « reseau », s'il appuie sur s'inscrire, celui-ci arrive sur la page principalement de l'application.
Grâce aux inputType, un clavier adapté à chaque champ s'affiche lorsqu'on clique sur la plage (un clavier numérique s'affiche quand on appuie sur « numéro de téléphone » par exemple)

Il serait nécessaire d'améliorer cette partie en mettant les champs obligatoires avant de changer de page.

Interface réinitialisation du mot de passe

Le fichier xml « reinitialisationmdp.xml » est associé aux Classe Java « MDP » et « gestionEmail ». Lorsque l'utilisateur est sur cette classe, il doit rentrer son e-mail puis appuyer sur le bouton « ENVOYER UN NOUVEAU MOT DE PASSE ».

La classe « gestionEmail » utilise un serveur SMTP pour se connecter à la boîte gmail de EMAmi : emami.nepasrepondre@gmail.com en donnant le mot de passe de celle-ci.

Ce serveur permet d'envoyer un e-mail à l'adresse préalablement remplie avec un mot de passe aléatoire généré dans la classe « MDP » dans la méthode : generateNewPassword()

Cette partie n'est pas optimale et se doit d'être améliorée. En effet la version de l'implémentation pour le serveur smtp n'est pas fonctionnelle et il a été difficile de trouver une version fonctionnelle. De plus, la création du mot de passe n'est pas très sécurisée car on peut obtenir deux mots de passes identiques mais aussi sans minuscule ou majuscule.

b. Interface centrale – Gautier Vasse

L'interface centrale est composée de 3 activités, une pour voir les amis avec qui nous nous sommes mutuellement acceptés, donc leur profil avec leurs réseaux sociaux apparaitront. Une autre activité avec la possibilité de voir les personnes en temps réel proche de nous avec la possibilité de demander une amitié immédiatement. Et enfin la dernière page recense les potentiels amis rencontrés durant les dernières 24h. L'interface est proche de celle de Tinder et Facebook avec une navigation possible entre les pages grâce à des icones en bas-de-page. Pour l'activité de perception des profils en temps réel, à l'arrière-plan nous faisons un scan pour détecter les appareils Bluetooth et en même temps nous enregistrons leur profil pour les mettre potentiellement dans la page des "vus ces dernières 24h".

c. Interface profil – Leelou Lebrun

La page profil fait partie des pages déroulantes de l'application. Permettant d'afficher les informations des utilisateurs pour les autres utilisateurs, elle est composée de trois modules différents. Pour que ces différents modules se suivent en déroulant, il faut utiliser un RecyclerView.

La page est donc créée à partir de six différents fichiers

- Fichier XML pour la photo et le nom et prénom
- Fichier XML pour le numéro de téléphone
- Fichier XML des noms des profils sur les différents réseaux et leur logo
- Fichier XML de l'organisation générale de la page avec le RecyclerView
- Classe Java RecyclerViewProfil qui crée la page
- Classe Java ActivitéProfil qui lance la page

d. Interface paramètres – Leelou Lebrun

La partie paramètres permet aux utilisateurs de changer leurs informations, en les supprimant, modifiants ou en en ajoutant. Elle est composée d'une page centrale qui est la seconde page déroulante de l'application. Elle se présente de manière similaire à la page profil, à l'exception qu'elle présente toutes les informations accompagnées de CheckBox qui permettent à l'utilisateur de choisir d'afficher ou non au public ces informations. Seuls le nom, prénom et photos ne peuvent être cachées. C'est en cliquant sur chacune des informations que l'on peut accéder à la page de modification.

Cette partie est donc composée de différents fichiers :

- Fichier XML pour les boutons photo et nom prénom
- Fichier XML pour le numéro de téléphone
- Fichier XML pour les différents réseaux
- Fichier XML pour l'ajout de réseaux
- Fichier XML de l'organisation générale de la page centrale avec le RecyclerView
- Classe Java RecyclerViewParam qui crée la page centrale

- Classe Java ActvitéParam qui lance la page centrale
- Les fichiers XML de toutes les pages de changement des informations
- Les classe Java des Activités qui permettent de lancer d-ces différentes pages

Approfondissement sur les classes RecyclerViewParam et RecyclerViewProfil

Ces classes sont très similaires, nous allons donc expliquer leur fonctionnement dans le même temps.
Les deux classes héritent de RecyclerView .adapter et sont fondamentales pour afficher une liste déroulante de différentes vues. Elles sont composées de différentes méthodes dont on expliquera rapidement l'utilité :

Le constructeur
Initialise le contexte, l'utilisateur et la liste des réseaux à afficher. Il lui faut pour cela communiquer avec la base de données pour aller chercher la liste des réseaux que l'utilisateur a décidé d'afficher (dans le cas du profil) ou a entré dans la base de données (dans le cas des paramètres).

Méthode getItemCount
Cette méthode retourne le nombre total d'éléments à afficher dans le RecyclerView.

Méthode getItemViewType
Cette méthode retourne le type de vue pour une position donnée. Cela crée les ItemView.

Les Classes ViewHolder
Chaque classe ViewHolder est utilisée pour aller chercher les composants des différentes vues du RecyclerView :
ViewHolderHaut gère la vue du haut avec le nom et la photo de l'utilisateur.
ViewHolderTel gère la vue avec le numéro de téléphone de l'utilisateur.
ViewHolderReseau gère les vues des réseaux sociaux de l'utilisateur.
ViewHolderAdd gère la vue pour ajouter un nouveau réseau.

Grâce à ces classes, on vient chercher les différents objets à mettre dans le layout si on a besoin de leur associé une valeur particulière à chaque profil.
Elles permettent également (dans le cas de la page de paramètres) d'assigner aux boutons un OnClick event qui permet de commencer une nouvelle activité qui ouvrira une nouvelle page (dans ce cas la page qui servira à changer les informations du compte).

Méthode onBindViewHolder
Cette méthode assigne les données particulières (la liste des réseaux et le numéro de téléphone notamment) à la vue, en fonction de la position et du type de vue. Elle gère la configuration des textes, des images et des écouteurs de clics.

Dans le cas des paramètres, cette méthode assigne également aux CheckBox leur fonction : Le réseau est affiché sur le profil si la case est cochée, sinon non.

Méthode onCreateViewHolder
Cette méthode gonfle (inflate) le layout appropriée en fonction du Viewtype et retourne le ViewHolder correspondant.

2) Base de données – Maxence Péron

La gestion des données est un aspect crucial de l'application, assurant la persistance, la cohérence et l'intégrité des informations stockées. Cette section du rapport détaille la conception et le fonctionnement du système de gestion de base de données de l'application EMAMI.

a. Conception de la base de données

La base de données locale de l'application EMAMI est construite en utilisant SQLite, un moteur de base de données léger et intégré directement dans Android. La conception de la base de données comprend trois principales tables : *RESEAU*, *COMPTE_RESEAU*, et *USER_OFFLINE*. Ces tables sont interconnectées pour refléter les relations entre les réseaux sociaux, les comptes utilisateur, et les utilisateurs eux-mêmes.

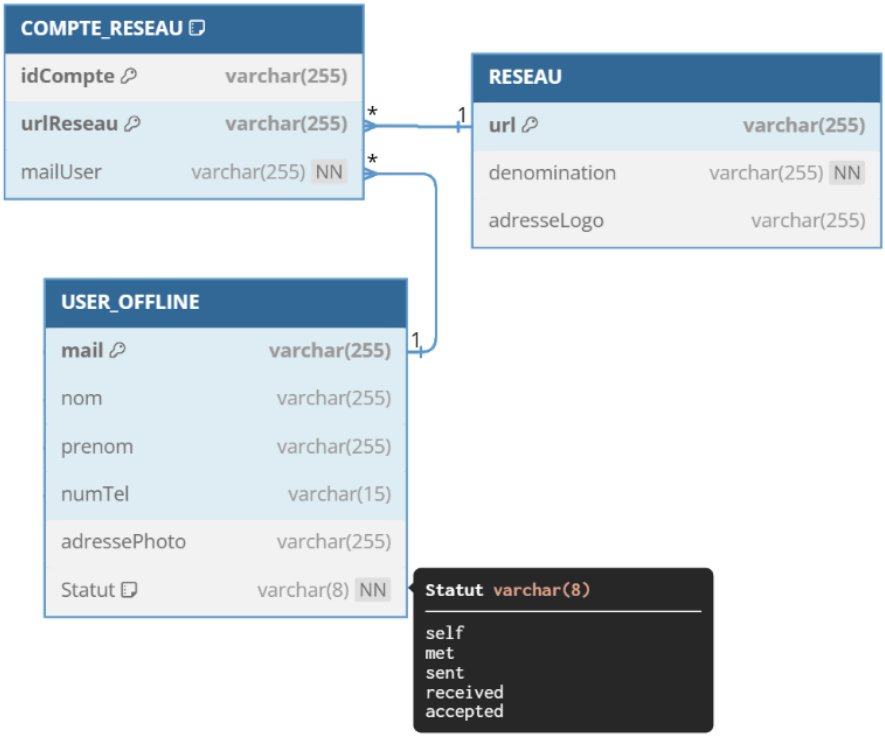


Figure 1 : Schéma de la base de données locale

Table *RESEAU*
url : URL du réseau social (clé primaire).
denomination : Nom du réseau social (unique et non nul).
adresseLogo : Adresse du logo du réseau social.

Table *COMPTE_RESEAU*
idCompte : Identifiant unique du compte (clé primaire).
urlReseau : URL du réseau social associé (clé étrangère).
mailUser : Adresse email de l'utilisateur EMAMI (clé étrangère, non nul).

Table *USER_OFFLINE*
mail : Adresse email de l'utilisateur (clé primaire).
nom : Nom de l'utilisateur.
prenom : Prénom de l'utilisateur.
numTel : Numéro de téléphone de l'utilisateur.
adressePhoto : Adresse de la photo de profil (unique).
Statut : Statut de l'utilisateur (valeurs possibles : self, met, sent, received, accepted).

b. Gestion des données

La gestion des données dans l'application EMAMI est réalisée à travers plusieurs classes Java qui encapsulent les opérations CRUD (*Create*, *Read*, *Update*, *Delete*) pour chaque table de la base de données. Ces classes assurent l'interaction avec la base de données de manière sécurisée et efficace.

Classes de gestion de la base de données

AppDatabase
Cette classe étend *SQLiteOpenHelper* et est responsable de la création et de la mise à jour de la base de données. Elle contient les requêtes SQL pour la création des tables et gère la connexion à la base de données.

ReseauBDD
Cette classe gère les opérations CRUD pour la table *RESEAU*.
Elle utilise des méthodes telles que *insertReseau*, *updateReseau*, *removeReseauWithURL*, et *getReseauWithURL*.

CompteReseauBDD
Cette classe gère les opérations CRUD pour la table *COMPTE_RESEAU*.
Elle inclut des méthodes telles que *insertCompteReseau*, *updateCompteReseau*, *removeCompteReseauWithIdCompte*, et *getCompteReseauWithIdCompte*.

UserBDD
Cette classe gère les opérations CRUD pour la table USER_OFFLINE.

Elle propose des méthodes comme *insertUser*, *updateUser*, *removeUserWithMail*, et *getUserWithMail*.

Le choix a été fait de n'utiliser que des clés primaires naturelles. L'utilisation de clés primaires artificielles pourrait cependant permettre d'améliorer la sécurité des données et de simplifier la modification d'entrées.

Validation des données et gestion des exception

Pour garantir l'intégrité des données et prévenir les erreurs, des validations sont mises en place avant chaque opération de base de données. Par exemple, avant d'insérer un nouveau réseau social, le système vérifie que les champs obligatoires ne sont pas nuls et respectent les contraintes définies.

De plus, des blocs try-catch sont utilisés pour gérer les exceptions SQL, assurant ainsi que l'application ne plante pas en cas d'erreur de base de données. Les erreurs sont loguées pour faciliter le débogage et l'amélioration continue de l'application.

Sécurité des mots de passe

Si nous avions eu plus de temps, nous aurions intégré un système de sécurisation des comptes utilisateurs. Il est prévu que l'application EMAMI implémente un système de gestion des mots de passe basé sur le hachage avec sel (*salt*). Ce procédé améliore la robustesse du chiffrement des mots de passe, rendant plus difficile les tentatives de piratage par attaques par dictionnaire ou par force brute.

Principe du hachage avec sel

Génération d'un Sel Unique : Lors de la création d'un compte utilisateur, un sel unique est généré pour chaque utilisateur. Ce sel est une valeur aléatoire qui est ajoutée au mot de passe avant le hachage.
Hachage du Mot de Passe : Le mot de passe combiné au sel est ensuite haché en utilisant un algorithme de hachage sécurisé (par exemple, SHA-256). Le résultat est stocké dans la base de données.
Vérification du Mot de Passe : Lors de la connexion, le mot de passe fourni par l'utilisateur est combiné avec le sel stocké, puis haché. Le résultat est comparé au hachage stocké dans la base de données. Si les deux valeurs correspondent, l'authentification est réussie.

c. Intégration d'une base de données serveur

Bien que la base de données locale soit essentielle pour l'opération hors ligne de l'application, une extension vers une base de données serveur est également prévue. Cette intégration permettra de gérer les requêtes de demande d'amis EMAMI.

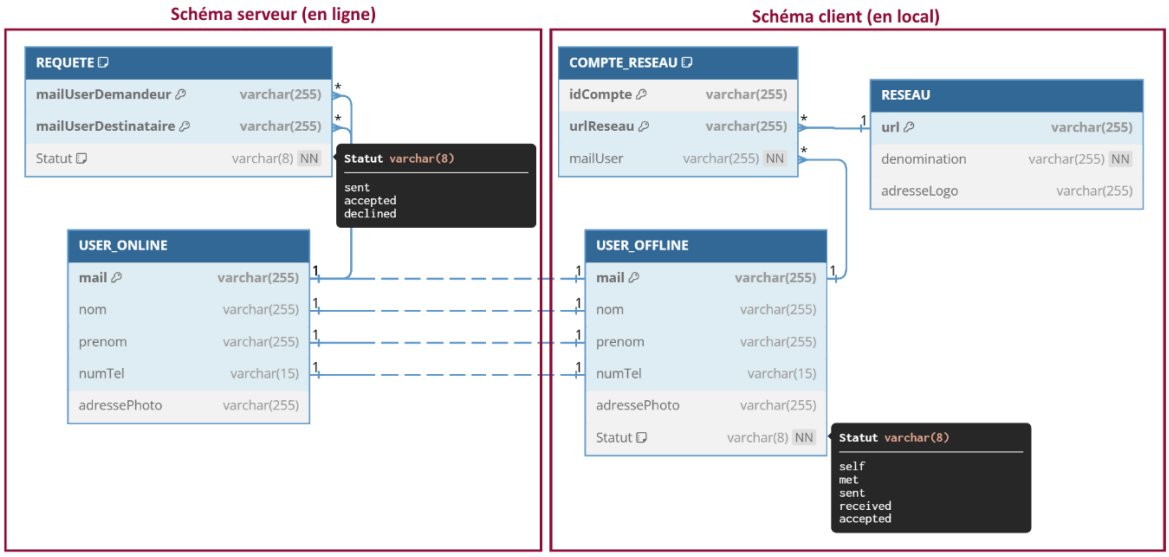


Figure 2 : Schéma des bases de données utilisées par EMAMI

Table *REQUETE*
mailUserDemandeur : Adresse email de l'utilisateur demandeur (clé primaire).
mailUserDestinataire : Adresse email de l'utilisateur destinataire (clé primaire).
statut : Statut de la requête (valeurs possibles : *sent*, *accepted*, *declined*).

Table *USER_ONLINE*
mail : Adresse email de l'utilisateur (clé primaire).
nom : Nom de l'utilisateur.
prenom : Prénom de l'utilisateur.
numTel : Numéro de téléphone de l'utilisateur.
adressePhoto : Adresse de la photo de profil (unique).

3) Bluetooth – Gaspard Langlais, Gautier Vasse

Dans notre application la partie en lien avec le Bluetooth est particulièrement importante, en effet c'est elle qui va permettre aux utilisateurs d'échanger certaines données sommaires entre eux. Cette partie a été source de nombreuse complication et c'est pourquoi elle n'est pas complétement implémentée dans le projet disponible sur GitHub.
De plus je n'ai pas fait de commit en mon nom sur le projet git, les tests étant fait de mon côté et étant relativement indépendant du reste, les seuls commit en rapport avec cette partie ci ont donc été réalisé par Gauthier.

a. Scenario d'usage

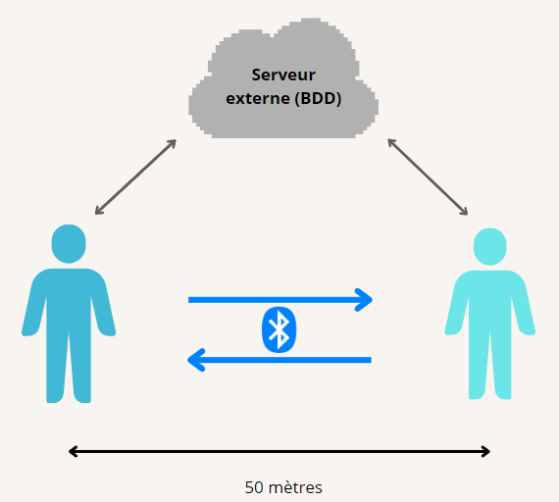
De manière à clairement comprendre à quoi va servir la technologie Bluetooth dans notre cas il est important d'analyser dans son ensemble la structure du projet via

ce scenario d'usage. A noter que nous verrons à la fin de cette partie qu'un autre scenario d'usage a été évoqué vers la fin du projet ce qui pourrait résoudre certaines complications que nous allons aborder.

Le scenario est alors le suivant :

- Deux utilisateurs avec l'application se rencontrent.
- Les téléphones s'échangent en Bluetooth leur id (adresse mail) ainsi que le nom et le prénom de l'utilisateur.
- Par la suite l'application va chercher sur la base de données l'image du profil.
- L'utilisateur accepte ou non de rentrer en contact avec la personne rencontrée.
- Si les deux acceptent l'application télécharge toutes les informations restantes à partir de la BDD.

Nous pouvons résumer ce scenario par le schéma suivant :



b. Choix de la technologie Bluetooth

La technologie Bluetooth est relativement ancienne et a connue de nombreuses variations, c'est pourquoi nous nous retrouvons avec différentes technologies utilisables sur nos smartphones. Il sera alors intéressant de comparer entre elles ces deux alternatives pour déterminée laquelle est la plus pertinente dans notre cas.

BLUETOOTH CLASSIQUE	BLUETOOTH LOW ENERGIE (BLE)
• Grande quantité de donnés (possible d'envoyer des images) . • Données en continue (idéal pour la musique). • Energivor. • Portée de 10 à 15 m.	• Petite quantité des données (pas d'images). • Flux ponctuel de données. • Basse consommation énergétique. • Portée jusqu'à 100 m. • Nécessite l'autorisation de position ainsi que la position activée.

Dans notre cas nous prévoyons une utilisation du Bluetooth de manière **continue** (en tache de fond même si l'application n'est pas aux premier plan). De plus nous n'échangeons que de **petits paquets de données** composés de texte. Enfin nous avons tout intérêt à privilégier une **portée importante** puisqu'en ville avec de nombreux murs/obstacle celle-ci se trouvera forcément restreinte.

C'est pourquoi nous nous tournerons donc vers une technologie **de Bluetooth Low Energy (BLE)**. Néanmoins de nombreux tests ont été pratiqué sur du Bluetooth classique, celui-ci étant bien plus documenté et accessible. En effet le BLE étant initialement fait pour permettre l'échange de données entre des appareils connectés et des machines classiques (téléphones, ordinateurs etc..) il en résultera alors certaines complications que nous allons abordés par la suite.

c. Initialisation du Bluetooth

Dans un premier temps il va être nécessaire de faire les demandes d'autorisation nécessaire, la liste précise des autorisations est la suivante :

- BLUETOOTH_SCAN
- BLUETOOTH_ADMIN
- ACCESS_FINE_LOCATION
- ACCESS_COARSE_LOCATION
- BLUETOOTH_CONNECT

On peut alors observer que des autorisations de positions sont demandées ce qui peut paraitre étonnant mais c'est absolument nécessaire au bon fonctionnement du BLE. Et c'est ici qu'apparait l'un des premiers problèmes liés à l'utilisation de cette technologie, en effet puisque le BLE peut permettre de géolocaliser les individus entre eux Android demande depuis toujours d'avoir des autorisations de positions.
Néanmoins depuis quelques années en plus de ces autorisations Android demande l'activation permanente de la position lors de l'utilisation du BLE, une mesure que nous devrons alors aussi respecter.

Pour finir nous allons initialiser les différents composants comme ceci :

- Dans un premier temps on vérifie si le Bluetooth est disponible, activé ...
- On crée un adaptateur pour interagir avec le bluetooth via: *getDefaultAdapter*
- L'initialisation de celui-ci se fait via : *getBluetoothLeScanner*

d. Scan des appareils

Le scan des appareils est une partie primordiale, en effet celui-ci doit permettre de recenser facilement les appareils autour de l'utilisateur et ce en un temps restreint, tout en ne commettant aucun oubli.
Pour ce faire la première étape est de définir une fréquence de scan, en effet dans le cas des technologies Bluetooth (et particulièrement pour le BLE) nous ne pouvons pas

réaliser de scan en parallèle les uns des autres et si un nouveau scan est relancé avant que le précèdent ne soit terminé nous risquons de perdre des données. Il y a donc un réel enjeu sur le choix de cette fréquence que nous avons dans notre projet fixé à un scan toutes les 2,5 secondes.

Ensuite pour ce qui est du scan en lui-même la technologie BLE nous mache le travaille puisque l'intégralité du scan peut être lancé via un simple appel de la méthode *startScan()*.

Enfin la dernière phase du scan consisté à récupérer la liste des adresses mac des appareils scanné ainsi que leur nom (si il a été définie). Nous réalisons cela grâce à la méthode *onReceive* qui nous permet d'obtenir un objet *device*. Cet objet device contient toutes les informations qui nous sont nécessaire, on peut y accéder via les méthodes *getName* ou *getAdresse*.

e. Connexion et échange de données

Nous avions abordé plus tôt dans le rapport le fait que la construction même du BLE (très orienté domotique) pouvait poser de nombreux problèmes et bien c'est à cette étape en particulier que cela va être le plus visible. En effet le BLE à été créer en supposant que l'appareil connecté jouerait un rôle de serveur (ici un serveur GATT) puisqu'il est le seul à fournir des données et que le téléphone jouerait le rôle de client.
Dans notre cas ce postulat est faux puisque nous voulons que les deux téléphones puissent à la fois envoyer et recevoir des données. Pour résoudre ce problème nous avons instauré un système de roulement, toutes les 10 secondes les appareils changent de rôle passant de serveur à client et inversement.

Une fois ce problème réglé nous pouvons nous connecter un autre appareil via la méthode *connectGATT* appliquée à l'objet *device* (trouvé précédemment). A noter qu'il est important pour maintenir un bon fonctionnement du système de mettre à jour l'état de connexion de l'appareil et de se déconnecter à la fin de l'échange de données.
Le cas des déconnexions forcés (appareils éteint ou Bluetooth désactivé de force) peut donc poser de nombreux problèmes que nous n'avons pas eu le temps de traité dans notre cas.

Enfin on vérifie dans un premier temps que l'application est présente (et dans la bonne version) sur le téléphone cible grâce à *discoverServices*. On lit par la suite les données nécessaires via *readCharacteristic*.

V. Notices

1) Notice d’installation

a. Prérequis

Avant de commencer, assurez-vous que votre appareil mobile répond aux critères suivants :

- Système d'exploitation : Android 8.0 ou supérieur / iOS 12.0 ou supérieur
- Bluetooth activé
- Connexion internet (Wi-Fi ou données mobiles)

b. Téléchargement de l'application

Pour Android :
Ouvrez l'application Google Play Store sur votre appareil.
Dans la barre de recherche, tapez "EMAMI".
Sélectionnez l'application EMAMI parmi les résultats de recherche.
Appuyez sur "Installer" et attendez la fin du téléchargement et de l'installation.

Pour iOS :
Ouvrez l'application App Store sur votre appareil.
Dans la barre de recherche, tapez "EMAMI".
Sélectionnez l'application EMAMI parmi les résultats de recherche.
Appuyez sur "Obtenir", puis sur "Installer" et attendez la fin du téléchargement et de l'installation.

c. Première ouverture de l'application

Une fois l'installation terminée, ouvrez l'application EMAMI depuis votre écran d'accueil.
Acceptez les permissions demandées pour permettre à l'application d'accéder à votre Bluetooth et à votre localisation.
Suivez les instructions à l'écran pour créer un compte ou connectez-vous si vous avez déjà un compte.

2) Notice d’utilisation

Appuyer sur l’icône de l’application -> Créer un compte -> Rentrer les informations suivantes : Nom, Prénom, Numéro de téléphone, E-mail, Mot de Passe et Confirmer le mot de passe -> CONTINUER -> Rentrer ses réseaux sociaux -> S’INSCRIRE

On tombe sur la page d'accueil :

- Bouton en bas à droite : Les paramètres de son compte.
  Appuyer sur les checkbox pour sélectionner sur les réseaux à afficher à ses connaissances.
  Cliquer sur le numéro pour le modifier
  Cliquer sur « AJOUTER UN RÉSEAU » pour en rentrer un nouveau.

La base de données n'est à ce jour pas liée à l'application, son architecture est cependant consultable dans la branche « BDD » du repository GitHub.

VI. Tests

Pour les tests, nous avons testé l'application sur des Emulateurs proposés par Android Studio et aussi nos smartphones mis en "mode développeur". Ces tests nous ont permis de voir si l'appli était fonctionnelle et les différents bugs rencontrés pour corriger le code.

VII. Licence

Notre projet d'application EMAMI se base sur un modèle gratuit avec possibilité de passé à une offre « premium » pour débloquer plus de fonctionnalité. Dans cette optique là nous avons pensé qu'utiliser une License propriétaire serait le plus optimale. En effet garce à cette License nous conservons les pleins pouvoir sur notre application tout en évitant de permettre à nos concurrents de nous voler certaines de nos technologies.

Ce mode de fonctionnement est particulièrement répandu pour les applications mobiles et nous pouvons remarquer que les entreprises sur un segment similaire, comme Hapnn, utilise une stratégie analogue.

Pour accompagner cette License nous avons rédigé une première version succincte de ce que pourrait être nos End User License Agreement (EULA's) que voilà ci-dessous :

1. Licence d'utilisation
   Vous êtes autorisé à utiliser cette application sur tous les appareils et plateformes compatibles. Cette application est fournie gratuitement.

2. Restrictions
   Vous n'êtes pas autorisé à :
   - Modifier, altérer ou créer des œuvres dérivées de cette application
   - Redistribuer, vendre, louer ou sous-licencier cette application
   - Décompiler, désassembler ou désosser l'application

3. Distribution

Vous pouvez distribuer cette application, mais uniquement sous forme non modifiée et dans son intégralité. Toute distribution doit inclure la présente licence et tous les droits d'auteur et autres avis de propriété.

4.Propriété intellectuelle

Cette application, y compris tout le code, les images et autres ressources, reste la propriété exclusive de EMAMI.

VIII. Perspectives

Une idée nous est venue tardivement sur une méthode plus accessible pour nous, codeur, afin d'échanger des données en BLE entre 2 utilisateurs. Néanmoins celle-ci étant très récente elle ne sera présentée ici que comme une hypothèse n'aillant jamais été vérifiée.
Cette méthode consiste à se passer de la phase d'échange de données, phase plus complexe et aboutissant à un grand nombre de bug, en stockant directement dans la base de données l'adresse Bluetooth de l'utilisateur (associé ainsi à son compte). Grace à cette méthode il nous suffit de réaliser un scan des appareils à proximité, de regarder si leur adresse se trouve dans la base de données et le cas échéant de lancer la procédure d'échange de données.

Cette méthode bien que pouvant aboutir à certaines faillent de sécurité (nécessité d'encoder l'adresse etc...) semble optimale.

Donc ce qu'il reste à faire dans l'application est :

- Sécuriser la base de données
- Améliorer l'UI et l'UX (comme toujours)
- Mettre en place la nouvelle méthode d'échange des données

EMAMI est une application prometteuse car on peut voir une réelle tendance sur les applications "courte distance" comme l'application mobile "Ten Ten" qui permet de parler à un ami comme avec un Talkie-Walkie. Cela montre une aisance d'avoir des liens à proximité par l'intermédiaire d'un smartphone.
Un des problèmes de ces applications c'est qu'elles sont TOR (tout ou rien) car les réseaux sociaux fonctionnent seulement s'il y a une grande base de données d'utilisateur. Il faudrait donc faire beaucoup de publicité au lancement de l'application ce qui requiert d'investir une somme monétaire bien trop importante pour des étudiants. Néanmoins, on pourrait la promouvoir par les réseaux sociaux en faisant des vidéos de promotion sur nos comptes personnels en essayant de faire le "buzz".
Notre équipe continue à développer le projet tout en faisant une analyse qualitative du marché pour estimer précisément le potentiel de l'application.