UE Projet

Documentation technique
BADHOC

badhoc

Réalisé par :

Marie AMARU

Mouncef NAJI

Roumaissa SOUKEHAL

Encadré par :

Abderrezak RACHEDI

Aurélien CHAMBON

Fonctionnalités

Communication par Bluetooth

L'application permet de communiquer par Bluetooth Low-Energy (BLE). Une fois connectés, les appareils peuvent communiquer par messagerie privée, ou par messagerie publique.

Réception de notifications par le serveur

Si l'appareil est déterminé comme dominant, il se connecte au serveur MQTT distant et lui envoie un message contenant des informations le concernant. En réponse, le serveur envoie un message contenant une notification, et le nœud dominant partage cette notification aux nœuds qui l'entourent, si une connexion Bluetooth a été établie au préalable avec ces derniers.

Envoi de message au serveur

Il est également possible pour les appareils d'envoyer des informations au serveur. Si l'appareil est dominant, il envoie le message directement au serveur, et s'il est dominé, il utilise l'appareil dominant comme relais pour envoyer le message au serveur.

Architecture technique

L'application est développée entièrement en Java et la version minimale du SDK Android est 21.

L'application est découpée en trois onglets :

- L'onglet **Around me** où apparaissent le nom des appareils alentour connectés par BLE et qui donne accès aux interfaces de messageries privées en cliquant sur le nom de l'un des appareils.
- L'onglet **Broadcast** qui regroupe les messages émis publiquement par les appareils à proximité.
- L'onglet **Notifications** qui regroupe les notifications reçues soit par le serveur directement si le nœud est dominant, soit relayées par le nœud dominant si l'appareil est dominé.

Initialisation du noeud

Au démarrage de l'application, le client Bridgefy est initialisé : le SDK récupère la licence passée par la clé d'API. Si le smartphone est connecté à Internet au premier démarrage, il se sert de la clé d'API passée dans le manifeste. Sinon, il se sert du certificat `com.igm.badhoc.txt` contenu dans les assets de l'application. Ce certificat est valable 4 semaines après la génération de la clé d'API. Après cette période, il faut se connecter impérativement à Internet au moins une fois pour que Bridgefy confirme la clé d'API.

Si l'enregistrement au service Bridgefy est réussi, deux évènements ont lieu.

Dans un premier temps, les listeners Bridefy sont initialisés : le listener MessageListener qui surveille la réception de messages, dont le fonctionnement est détaillé dans la section *Fonctionnalité de messagerie*, ainsi que le listener StateListener dont le but est de détecter de nouveaux appareils à proximité et d'envoyer le message handshake le cas échéant.

Il détecte également lorsque la connexion à un voisin est perdue, si ce dernier ferme l'application ou coupe sa connexion Bluetooth par exemple. Enfin il vérifie que les permissions de localisation qui sont obligatoires pour lancer le SDK Bridgefy ont bien été approuvées par l'utilisateur, sinon il retourne un message d'erreur.

Dans un second temps, le champ Node de l'activité principale est initialisé. Ce champ contient toutes les informations techniques de l'appareil :

- son nom, constitué du modèle de téléphone et de sa marque.
- son type : 1 pour smartphone.
- sa vitesse si elle peut être récupérée, sinon initialisée à 0 km/h.
- son adresse MAC si elle peut être récupérée.
- son RSSI.
- sa localisation : sa latitude et sa longitude.
- son signal LTE.
- son statut : par défaut, le statut de l'appareil est **dominé**, et une tâche récurrente est initialisée et répétée toutes les 10 secondes afin de surveiller l'évolution de ce statut. La logique de la tâche est la suivante :
    - Si l'appareil est connecté à Internet, que son statut est **dominé**, et qu'il n'a pas de nœud dominant dans son entourage (son champ dominant est null), alors il devient dominant. Il démarre le service pour se connecter au serveur, et envoie un message Broadcast avec

la payload `POTENTIAL_DOMINANT` qui prévient son entourage qu'il est dominant. La gestion de ce message par les autres nœuds est expliquée dans la section *Fonctionnalité de messagerie.*

- Si l'appareil n'est plus connecté à Internet et que son statut est **dominant**, il change son statut à **dominé** et arrête le service de connexion au serveur. Enfin il envoie un message Broadcast avec la payload `NO_LONGER_DOMINANT` qui prévient son entourage qu'il n'est plus dominant. La gestion de ce message est aussi expliquée dans la section *Fonctionnalité de messagerie.*

Etablissement de la connexion entre les appareils

Grâce au SDK Bridgefy, l'application Badhoc permet aux appareils équipés de l'application de se connecter par connexion BLE.

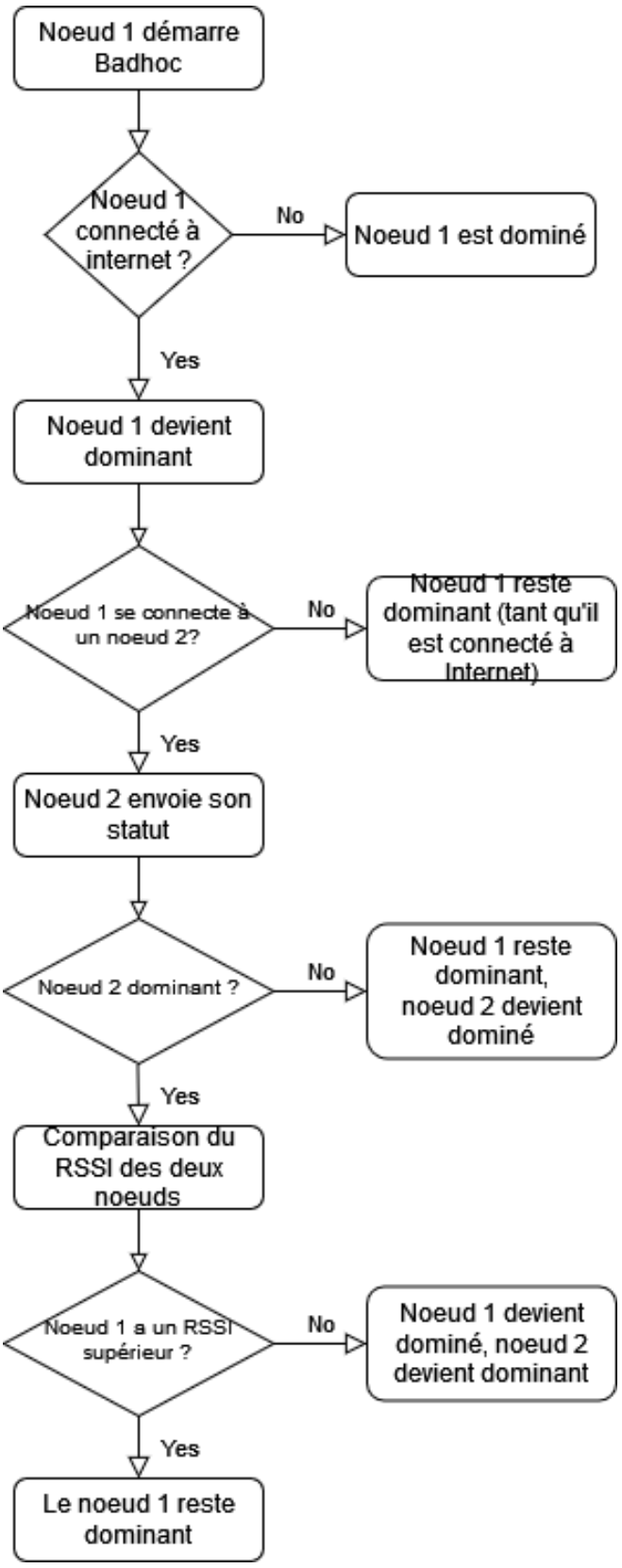
L'application scanne en permanence ses alentours si une connexion Bluetooth avec un autre appareil est disponible. Si un tel appareil est disponible, un message **handshake** lui est envoyé. Ce premier message contient des informations de base à propos de l'appareil (son nom, son adresse MAC, son RSSI, et s'il est dominant ou non). Toute cette gestion a lieu dans le listener `MessageListenerImpl`.

Lorsque l'appareil reçoit ce message handshake, il enregistre l'appareil voisin dans sa liste de voisins.

L'appareil est ajouté à la liste de nœuds voisins dans le fragment AroundMeFragment et apparaît dans l'onglet Around Me. Si la connexion est active, le logo à côté du nom de l'appareil est vert. Si la connexion est perdue, le logo devient noir.

Lors de ce premier contact, l'appareil actuel compare alors son statut avec celui du device voisin. La logique pour déterminer qui est le noeud dominant entre les deux est la suivante :

- Si le nœud actuel est dominant et que le nœud voisin est dominé, leurs statuts respectifs ne changent pas.
- Si le nœud actuel est dominé et que le nœud voisin est aussi dominé, leurs statuts respectifs ne changent pas.
- Si le nœud actuel est dominant et que le nœud voisin est aussi dominant, leur RSSI est comparé, et celui avec le meilleur signal devient dominant. Ses informations sont alors mises à jour : vider sa liste de nœuds dominés et lui assigner un dominant s'il perd son statut de dominant, ou ajouter le nœud voisin à sa liste de dominés et lui désassigner son dominant actuel s'il gagne le statut de dominant.
- Si le nœud actuel est dominé et que le nœud voisin est dominant, alors le nœud actuel est ajouté à la liste de nœuds dominés du nœud voisin et le nœud voisin devient le nœud dominant du nœud actuel.


Noeud 1 démarre Badhoc
Noeud 1 connecté à internet ?
No
Noeud 1 est dominé
Yes
Noeud 1 devient dominant
Noeud 1 se connecte à un noeud 2?
No
Noeud 1 reste dominant (tant qu'il est connecté à Internet)
Yes
Noeud 2 envoie son statut
Noeud 2 dominant ?
No
Noeud 1 reste dominant, noeud 2 devient dominé
Yes
Comparaison du RSSI des deux noeuds
Noeud 1 a un RSSI supérieur ?
No
Noeud 1 devient dominé, noeud 2 devient dominant
Yes
Le noeud 1 reste dominant

Lorsque ce premier handshake est réalisé, les deux appareils sont connectés et peuvent échanger des messages privés ou publics.
Le listener `MessageListenerImpl` reçoit par la suite les messages envoyés par les appareils connectés.

Deux méthodes séparent les messages reçus de type privé ou de type Broadcast.

Si le message reçu est privé, et ne contient pas de payload avec le nom de l'appareil, cela signifie qu'il ne s'agit pas d'un message handshake et que le destinataire est déjà connu. Une partie `PAYLOAD_PRIVATE_TYPE` de la payload du message est analysée, et le message privé reçu peut être de trois types :

- `TEXT` : il s'agit d'un message de type privé régulier, qui contient uniquement du texte. Le message est alors ajouté à la liste de messages privés du fragment PrivateChatFragment, associé à l'id de l'appareil qui a envoyé le message. Cela permet de mettre à jour la liste des messages du fragment PrivateChatFragment et de faire apparaître le nouveau message dans l'interface.
- `IMAGE` : il s'agit d'un message de type privé qui contient une image. De la même façon, le message est ajouté à la liste de messages privés du fragment PrivateChatFragment.
- `MESSAGE_TO_SERVER` : ce message privé ne peut être reçu que si le nœud est dominant. En effet il s'agit d'un message qu'un nœud dominé souhaite envoyer au serveur, et le nœud dominant sert de relais pour l'envoyer au serveur. Le contenu n'est pas affiché, il est simplement dirigé vers le service de connexion au serveur pour être envoyé.

Si le message reçu est de type Broadcast, alors une partie `PAYLOAD_BROADCAST_TYPE` de la payload du message est analysée, et le message Broadcast reçu peut être de quatre types :

- `REGULAR_BROADCAST` : il s'agit d'un message de type broadcast régulier, cela correspond à un message émis sur le channel public des communications et il sera ajouté au fragment BroadcastFragment qui permet de mettre à jour la liste de messages publics, et ainsi tous les appareils connectés verront ce nouveau message apparaître dans l'onglet Broadcast.
- `FROM_SERVER` : il s'agit d'un message provenant du nœud dominant du réseau qui partage la notification qu'il a reçu du serveur vers ses nœuds dominés. Le message est alors ajouté au fragment NotificationFragment et apparaît dans l'onglet Notifications.
- `NO_LONGER_DOMINANT` : il s'agit d'un message provenant du nœud dominant du réseau qui prévient qu'il a perdu son statut dominant pour cause de perte de connexion Internet. Ainsi les informations du nœud actuel sont mises à jour : on désassigne son nœud dominant.
- `POTENTIAL_DOMINANT`: il s'agit d'un message d'un nœud voisin qui prévient qu'il est désormais potentiellement dominant lorsque le nœud dominant du réseau a disparu. Si le nœud actuel est dominé, alors il enregistre son voisin comme dominant. Si il est lui aussi potentiellement dominant, alors leurs RSSI sont comparés à nouveau avant de décider qui est le nouveau nœud dominant.

Notifications push et badges de notification

Pour les appareils Android de version supérieure à Android 26, lorsque le service `ServerService` est démarré et qu'une connexion est établie avec le serveur, une notification push est affichée dans la barre déroulante du téléphone. Elle permet de prévenir l'utilisateur que l'application utilise sa connexion en fond lorsque l'application est en pause.

Ces appareils reçoivent également une notification push lorsque l'application est en pause et qu'un message privé ou public a été reçu.

Lorsque l'application est ouverte, tous les appareils voient un badge rouge apparaître à droite de l'onglet concerné lorsqu'un message privé ou public est reçu.

Envoi d'une image

L'envoi d'une image est possible dans le cadre de la messagerie privée. La taille de l'image est compressée afin de pouvoir être envoyée plus rapidement.

Réception de notifications du serveur

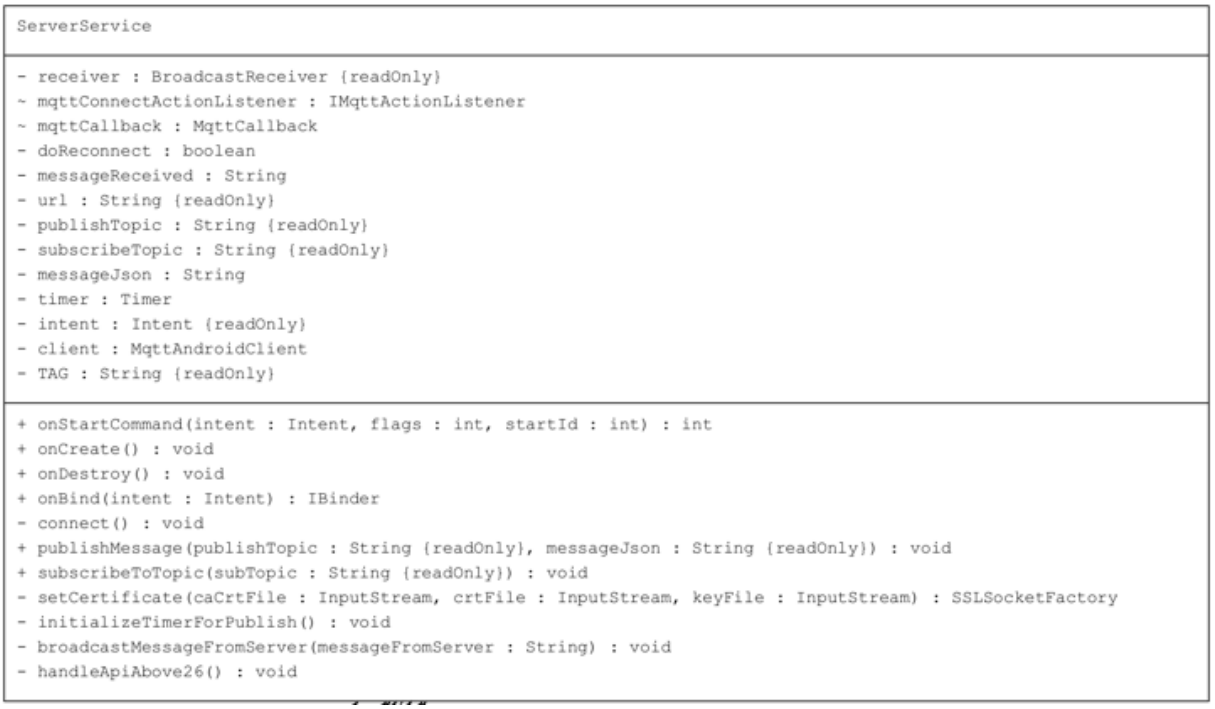
La connexion au serveur est implémentée dans la classe `ServerService`. Si un appareil est dominant, le service `ServerService` démarre. Ce service gère toutes les fonctionnalités liées au serveur. Il contient les différents listeners qui réagissent en fonction des événements : si la connexion est établie, si la connexion est perdue, si un message est publié sur un topic dans un sens ou dans l'autre.

Il gère également l'ajout du certificat de sécurité client, de la clé privée cliente et du certificat CA afin de pouvoir établir une connexion SSL avec le serveur distant.

Lorsque la connexion est établie, le service publie toutes les 30 secondes un message sur le topic **nodekeepalive** du serveur grâce à une tâche récurrente. Ce message est de la forme suivante :


```
{
  "type": "1",
  "speed": "55",
  "lteSignal": "-45",
  "isdominant": 1,
  "dominating": ["00:00:00:f0:bb:7c"]
  "macAddress": "00:00:00:d3:cf:15",
  "latitude": "5.000137167266086",
  "longitude": "4.999995411496197",
  "neighbours": [{"macAddress": "00:00:00:f0:bb:7c", "RSSI": -39.166666666665776}]
}
```


L'appareil est également abonné au topic **notifs** du serveur, qui lui envoie à chaque réception d'un message sur **nodekeepalive** une notification.
Lorsque le message du serveur est reçu, il est propagé vers le fragment NotificationFragment afin de mettre la vue à jour et d'ajouter la notification à la liste. Le message est également envoyé par message de type Broadcast vers les autres appareils voisins du nœud avec la payload `FROM_SERVER` afin de leur communiquer le message de notification et l"afficher dans l'onglet Notifications.



Ce service est démarré ou stoppé depuis l'activité principale en fonction du statut de l'appareil. S'il est dominant, le service est lancé en parallèle de l'activité principale. Si le nœud n'est plus dominant, alors le service est stoppé. Enfin, si le nœud est dominé, le service n'est pas démarré.

Envoi de message au serveur

Il est possible pour un appareil d'envoyer un message au serveur s'il est dominant ou s'il possède un appareil dominant dans son réseau.

Lorsque qu'il effectue l'action d'envoi de message dans le fragment NotificationFragment, deux situations sont possibles :

- Le nœud est dominant, alors il communique simplement avec le service pour publier son message sur le topic **nodekeepalive**. Le message est

ajouté à la liste de Notifications et la vue est mise à jour pour faire apparaître ce message envoyé.

- Le nœud est dominé, alors il se sert de son nœud dominant pour envoyer le message vers le serveur. Il envoie alors le message sous forme de message privé contenant la payload `MESSAGE_TO_SERVER` au nœud dominant du réseau, et ce dernier va propager ce message au serveur. Le message envoyé apparaît donc dans l'onglet Notification du nœud qui l'a envoyé, mais pas dans l'onglet du nœud dominant, qui ne s'intéresse pas au contenu de ce message.

Détails techniques de l'implémentation

Communication entre service et fragment de notification

Dans certaines situations, le service doit communiquer avec le fragment de notification. Un BroadcastReceiver est instancié dans le fragment NotificationFragment et surveille l'Intent du serveur `INTENT_SERVER_SERVICE` :

- Lorsqu'un message est reçu dans le topic **notifs**, et que le message doit être ajouté au fragment de notifications, on envoie un broadcast `ACTION_NOTIFICATION_RECEIVED`.
- Lorsque la connexion au serveur est perdue ou que le service est stoppé, il faut changer l'affichage du titre dans le fragment avec un broadcast `ACTION_CHANGE_TITLE`. Le service précise si les notifications reçues proviennent du serveur directement et indique qu'il faut mettre à jour le titre du fragment, ainsi que l'icône à côté. (`TITLE_DOMINANT` ou `TITLE_NOT_DOMINANT`).

Communication entre activité principale et fragment de notification

Lorsque l'activité principale souhaite démarrer ou stopper le service de communication avec le serveur en fonction du statut du nœud, il envoie un Intent `INTENT_MAIN_ACTIVITY` qui contient une payload `ACTION_CONNECT` qui indique au fragment s'il faut démarrer ou stopper le serveur.

Communication entre activité principale et service

Le service contient un champ `messageJson` représentant le contenu du message à publier sur le topic **nodekeepalive**. Ce champ est initialisé au démarrage du service à l'aide d'un extra passé à l'intent. Cependant ce champ doit être mis à jour à chaque nouveauté dans l'état du nœud actuel. Un BroadcastReceiver est instancié dans le service et surveille l'Intent de l'activité principale `INTENT_MAIN_ACTIVITY`:

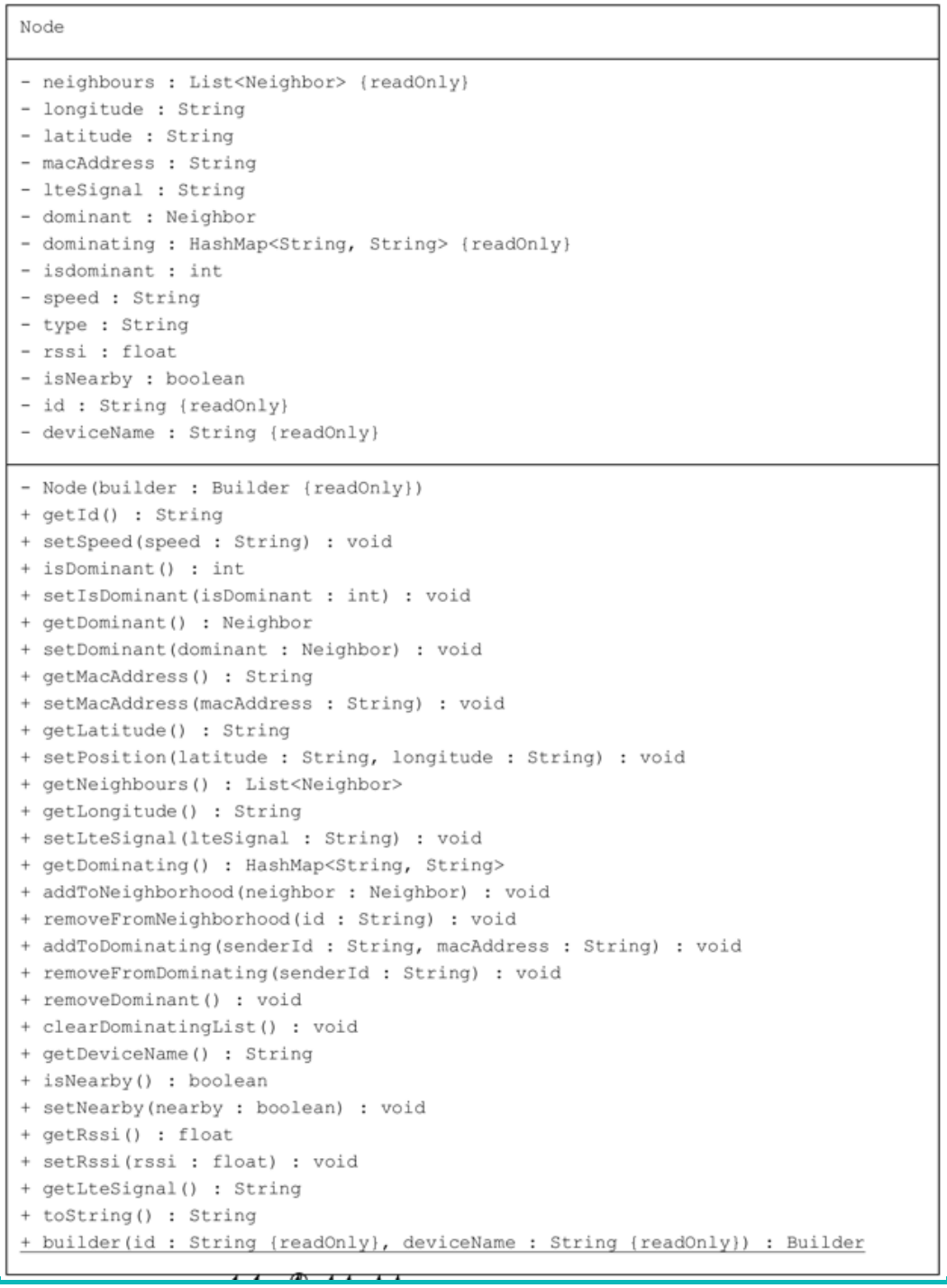
- Le champ **messageJson** est mis à jour grâce au contenu du broadcast `ACTION_UPDATE_NODE_INFO` envoyé par l'activité principale.

Entités de l'application (package model)

Le projet utilise plusieurs classes pour représenter les objets liés au fonctionnement de l'application. Nous détaillons ici leur contenu et leur utilité fonctionnelle.

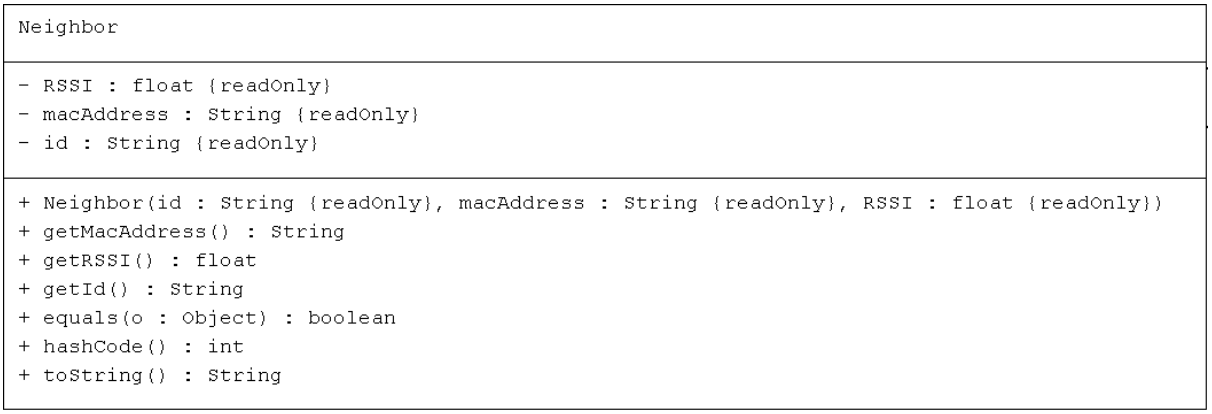
Node

Cet objet représente un nœud dans son format le plus complet.



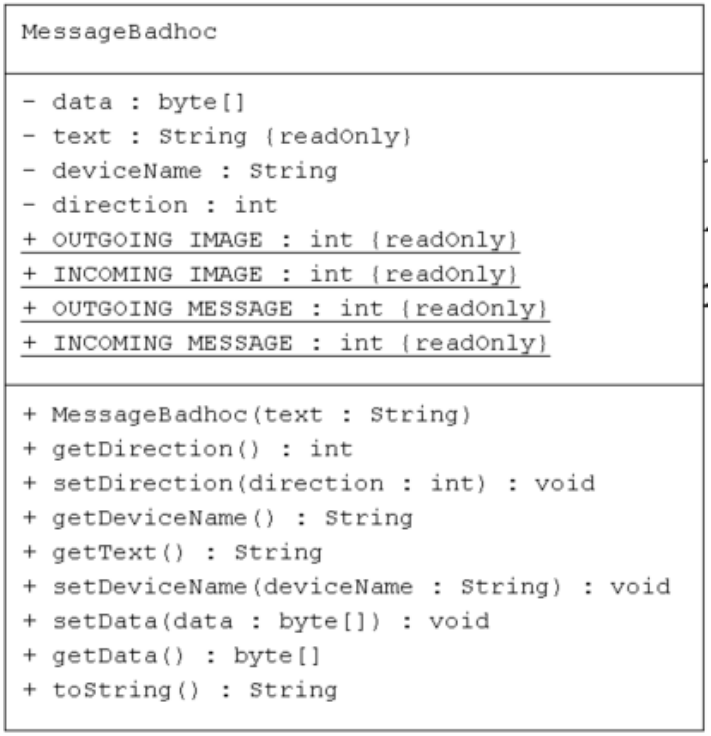
Neighbor

Cet objet représente également un nœud mais dans un format beaucoup plus simple, ne contenant que les informations pertinentes qu'un nœud doit avoir sur ses voisins proches.



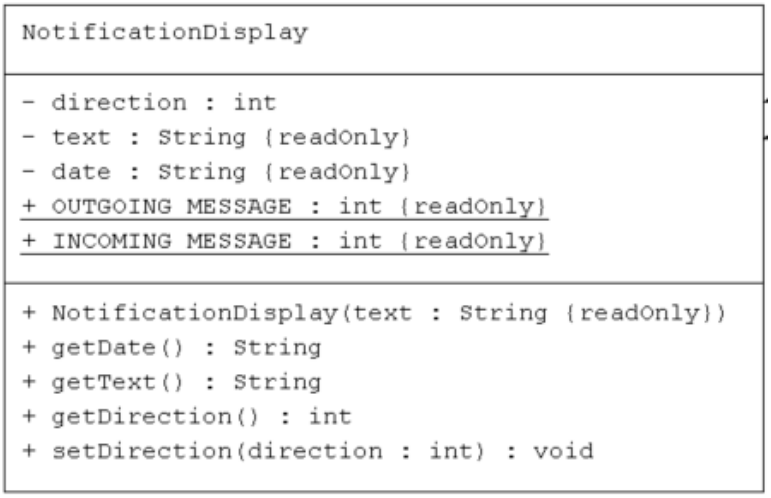
MessageBadhoc

Cet objet représente un message avec les informations nécessaires à l'affichage des messages dans les fragments de conversations (`BroadcastChatFragment` et `PrivateChatFragment`).



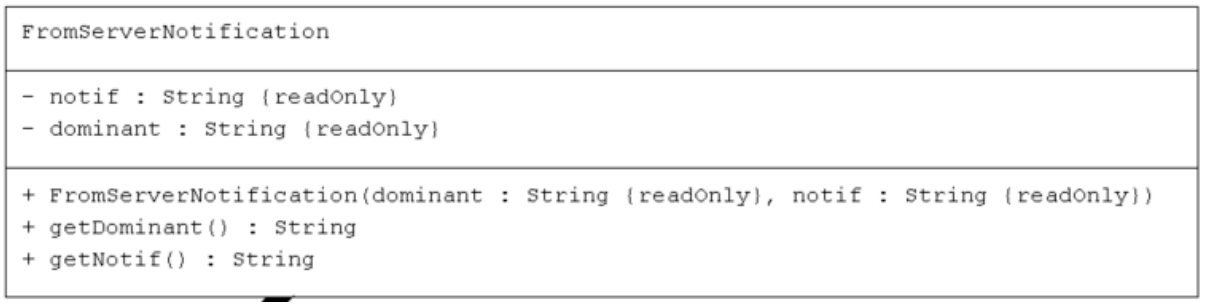
NotificationDisplay

Cet objet représente toutes sortes de notification en entrée ou en sortie. Il sert à afficher correctement le contenu de la notification dans le fragment `NotificationFragment`.



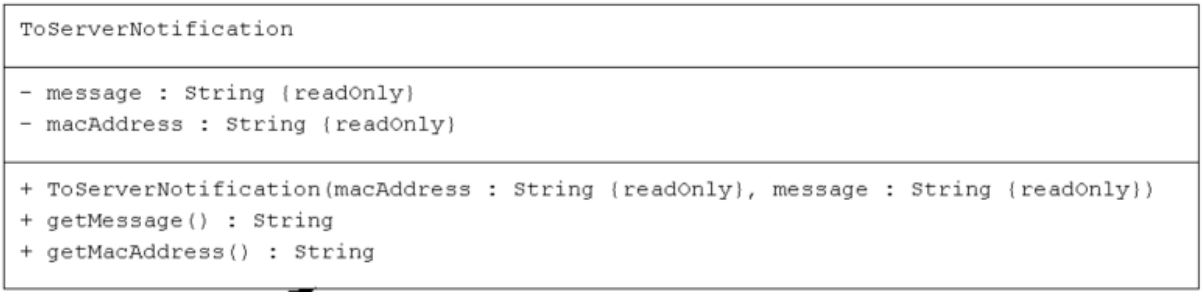
FromServerNotification

Cet objet représente une notification en provenance du serveur. Elle contient un champ dont le contenu est l'adresse MAC du nœud dominant du réseau, ainsi que le contenu de la notification.



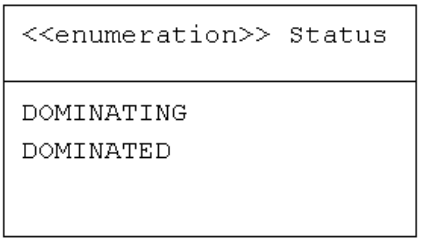
ToServerNotification

Cet objet représente une notification à destination du serveur. Elle contient un champ dont le contenu est l'adresse MAC du nœud dominant du réseau, ainsi que le contenu de la notification.



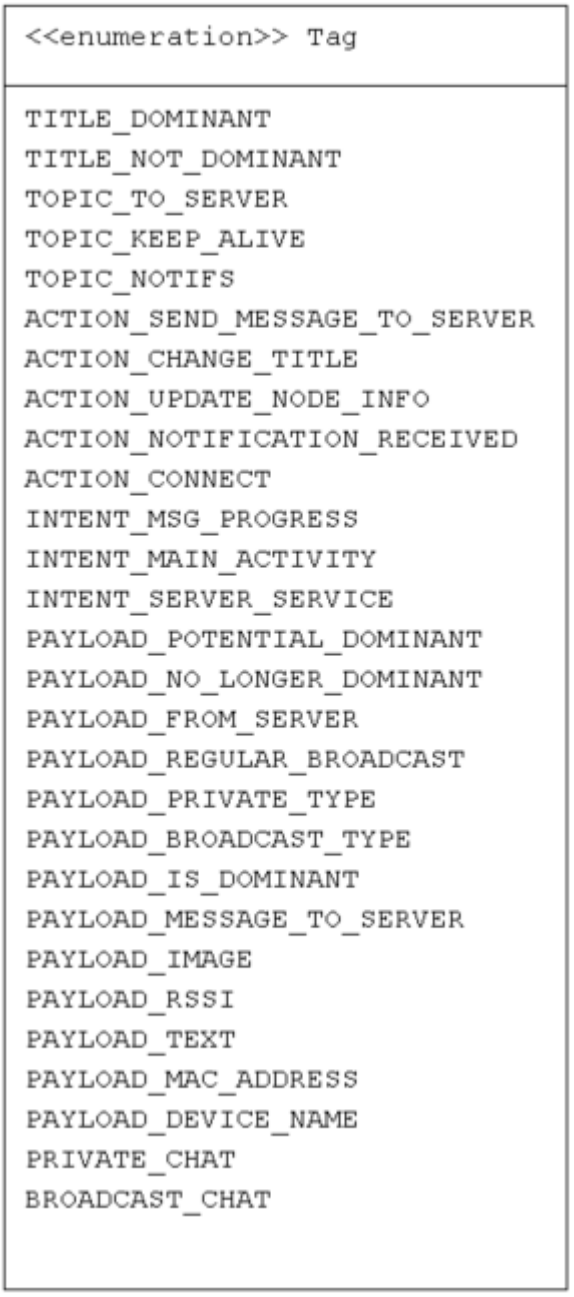
Status

Cette énumération représente les deux statuts qu'un nœud peut avoir : dominant ou dominé.



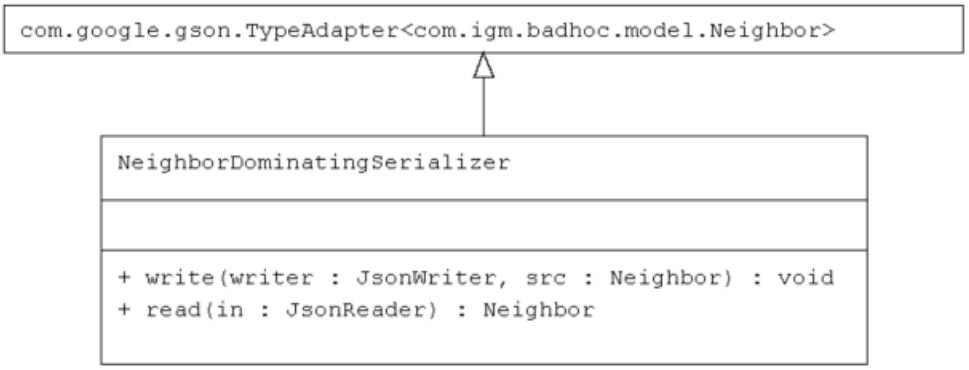
Tag

Cette énumération contient toutes les clés associées à des chaînes de caractère. Les clés sont liées soit à l'envoi de broadcast dans le contexte de la communication entre l'activité, les fragments et le service, soit dans les payloads personnalisées envoyées dans les messages privés ou publics entre les appareils.


<<enumeration>> Tag
TITLE_DOMINANT
TITLE_NOT_DOMINANT
TOPIC_TO_SERVER
TOPIC_KEEP_ALIVE
TOPIC_NOTIFS
ACTION_SEND_MESSAGE_TO_SERVER
ACTION_CHANGE_TITLE
ACTION_UPDATE_NODE_INFO
ACTION_NOTIFICATION_RECEIVED
ACTION_CONNECT
INTENT_MSG_PROGRESS
INTENT_MAIN_ACTIVITY
INTENT_SERVER_SERVICE
PAYLOAD_POTENTIAL_DOMINANT
PAYLOAD_NO_LONGER_DOMINANT
PAYLOAD_FROM_SERVER
PAYLOAD_REGULAR_BROADCAST
PAYLOAD_PRIVATE_TYPE
PAYLOAD_BROADCAST_TYPE
PAYLOAD_IS_DOMINANT
PAYLOAD_MESSAGE_TO_SERVER
PAYLOAD_IMAGE
PAYLOAD_RSSI
PAYLOAD_TEXT
PAYLOAD_MAC_ADDRESS
PAYLOAD_DEVICE_NAME
PRIVATE_CHAT
BROADCAST_CHAT

Les outils de sérialisation (package serializer)

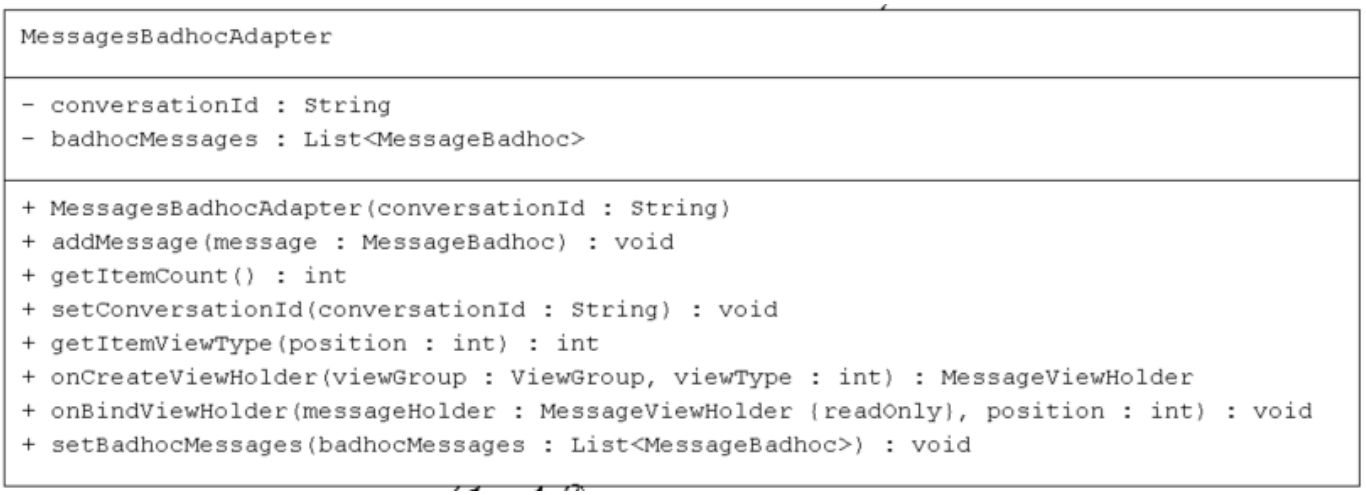
La classe **`NeighborDominatingAdapter`** sert à sérialiser au bon format le contenu de la classe **`Neighbor`** pour que le message au format JSON ait le format attendu par le serveur.



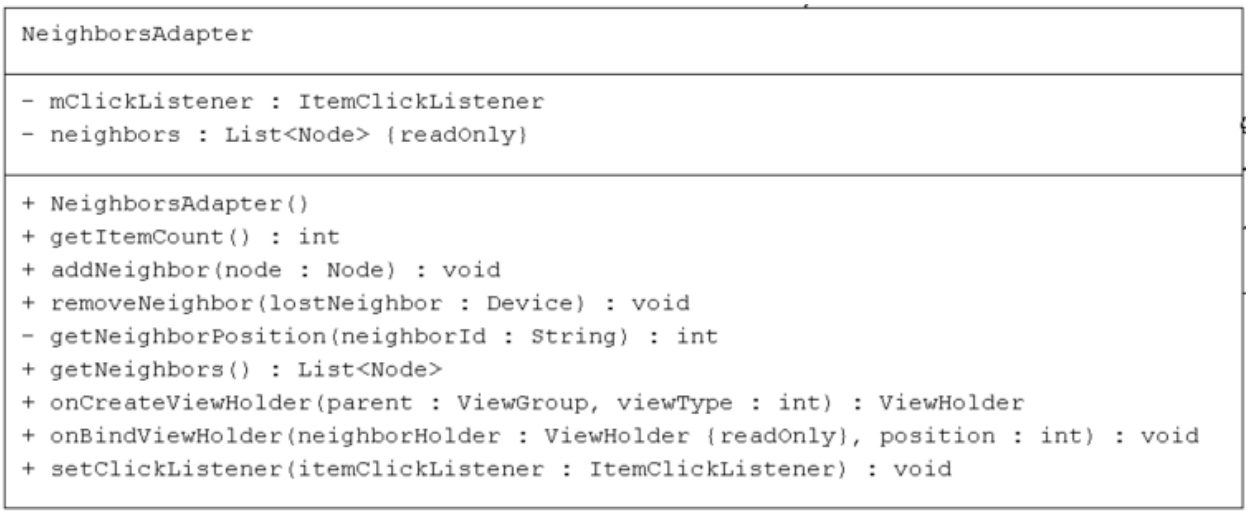
Les adapters (package adapter)

Les adapters définis dans le package **`adapter`** sont des adapters liés aux Recyclerview des fragments. Ils servent à gérer l'affichage du contenu des fragments.

L'adapter **`MessagesBadhocAdapter`** est partagé entre les deux fragments **`BroadcastChatFragment`** et **`PrivateChatFragment`** car l'affichage est le même : la liste des messages reçus et envoyés.



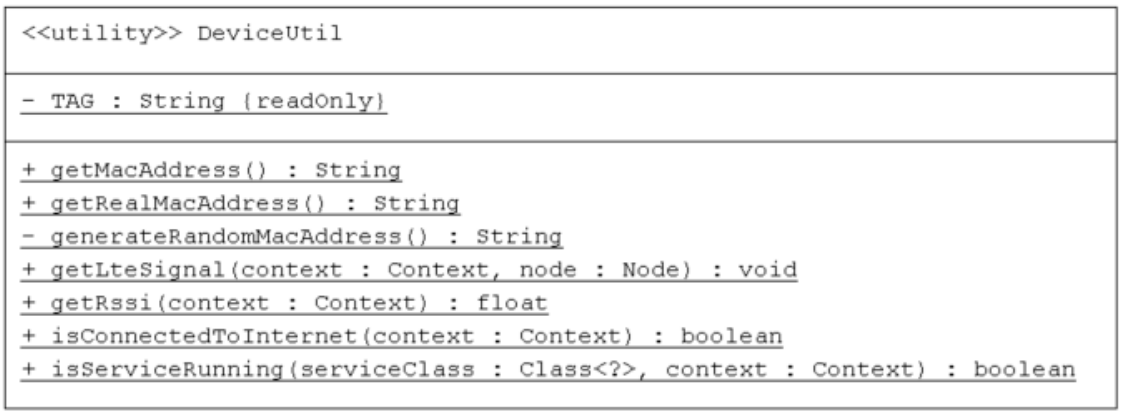
L'adapter `NeighborsAdapter` sert à afficher les appareils connectés dans l'onglet Around me dans une liste composée de leurs noms et d'une icône dont la couleur change s'ils sont connectés ou si la connexion avec eux a été perdue.



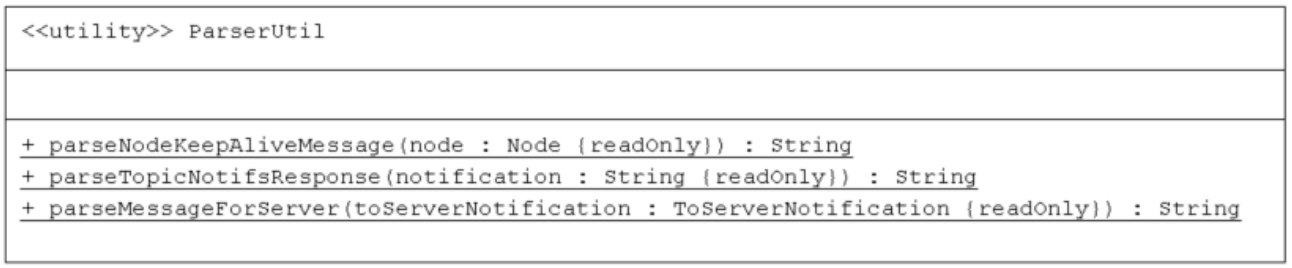
L'adapter `NotificationAdapter` gère l'affichage des notifications dans l'onglet Notifications.

Les outils (package util)

La classe `DeviceUtil` regroupe toutes les méthodes statiques qui permettent d'accéder aux informations techniques de l'appareil, telles que les adresses MAC, ou de les générer lorsqu'il n'est pas possible de les récupérer. Elle permet de récupérer le signal LTE, le signal RSSI, vérifier la connexion à Internet ou si un service Android a bien été démarré dans l'application.

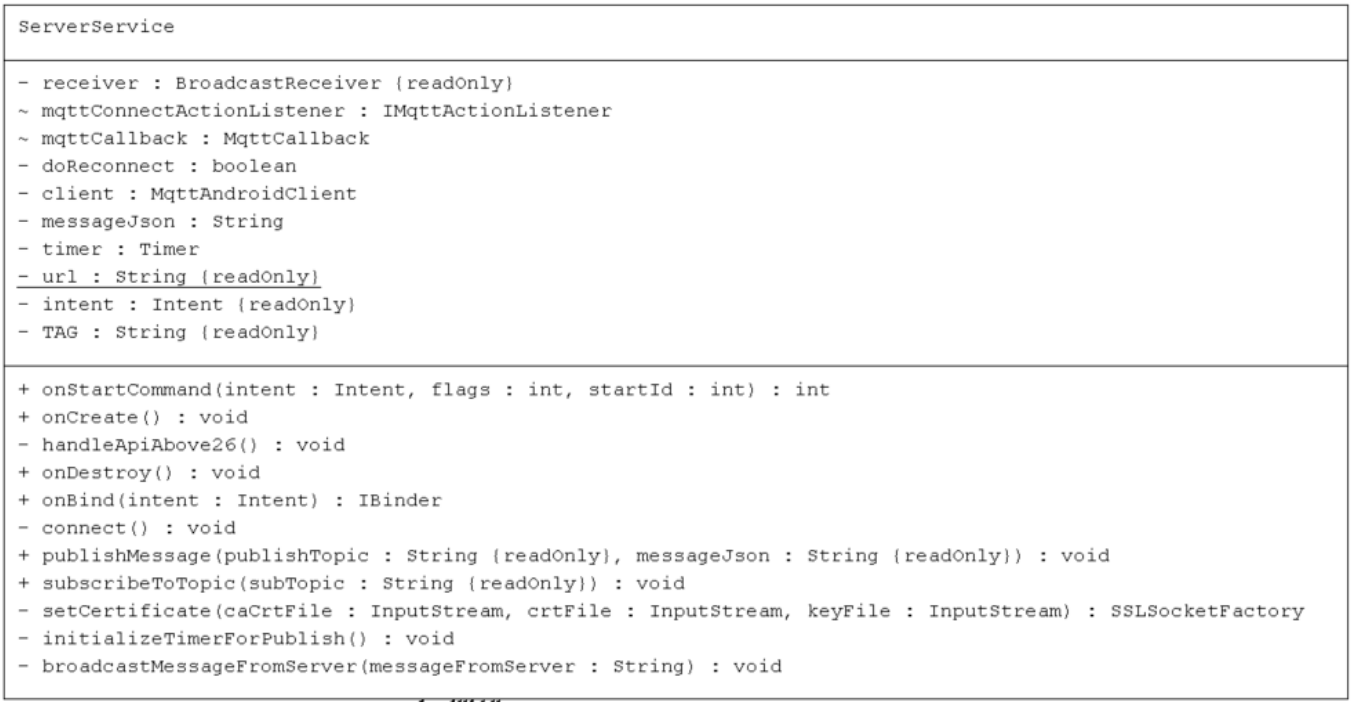


La classe ParserUtil regroupe toutes les méthodes statiques qui permettent d'utiliser la librairie Gson afin de formater des chaînes de caractères au format JSON, afin d'être envoyées et comprises par la suite par le serveur. On y trouve également des méthodes pour lire les messages du serveur, et les transformer en objets de l'application.

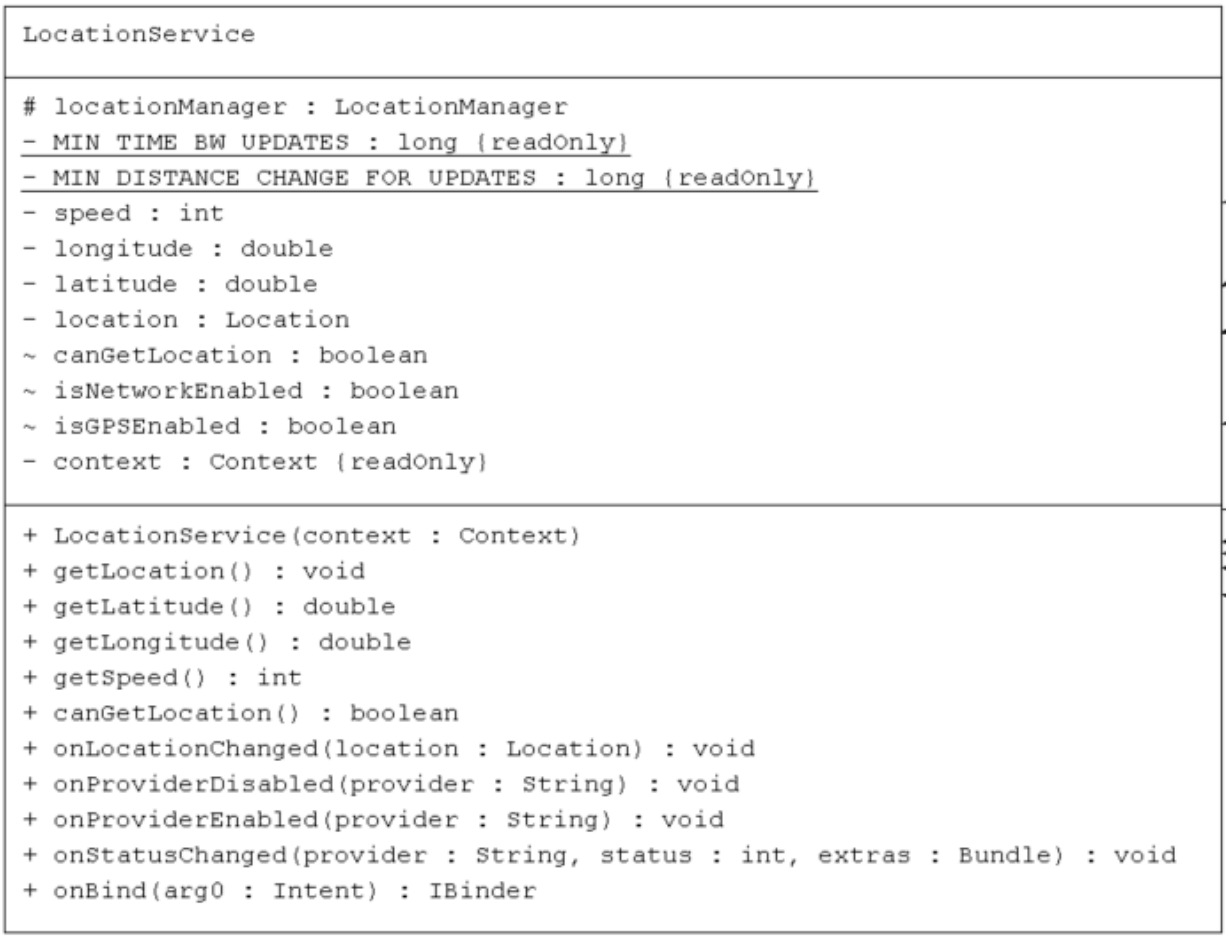


Les services (package service)

Le service `ServerService` sert à la gestion de la connexion au serveur. Il permet d'initialiser la connexion et de gérer la perte de connexion. C'est également dans ce service que se trouve la gestion des certificats afin de permettre la connexion SSL au serveur.



Le service `LocationService` sert à gérer la position et le déplacement de l'appareil. Grâce à ce service, on récupère la latitude et la longitude ainsi que la vitesse de l'appareil en utilisant soit les informations réseaux de l'appareil si elles sont disponibles, soit le GPS de l'appareil sinon afin d'obtenir les informations requises.



Tests unitaires et d'intégration

Les tests unitaires de l'application se trouvent dans le dossier test/java/com/igm/badhoc, tandis que les tests d'intégrations se situent dans le dossier androidTest/java/com/igm/badhoc.

Les tests unitaires couvrent toutes les classes qui pouvaient être testées sans nécessité d'utiliser un environnement Android. Ont donc été testées toutes les méthodes constructeur, les méthodes getter, les méthodes equals et hashcode lorsqu'elles ont été redéfinies. Les méthodes de la classe DeviceUtil ont également été testées lorsque c'était possible, mais ont été testées de façon plus approfondie dans les tests d'intégration.

Les méthodes de la classe ParserUtil ont été testées pour vérifier que les méthodes formaient des messages en sortie au bon format.

Les tests d'intégration, qui permettent de tester des méthodes dans l'environnement Android en démarrant un simulateur, ont permis de tester l'affichage des éléments. Chaque fragment contient des éléments (boutons, titres…) à afficher à leur démarrage , ce qui est testé dans ces tests. Le démarrage du service ServerService est également testé, ainsi que les méthodes de la classe DeviceUtil. Ces méthodes permettent de récupérer des informations liées à un appareil et nécessitent donc d'être testées sur un émulateur.

Workflow GitHub Actions

Afin d'assurer la non régression du développement, un script GitHub Actions qui lance les tests unitaires à chaque commit sur Git a été rédigé. Il permet de vérifier que tous les tests passaient correctement à chaque ajout de code.

Il se situe dans le dossier .github/workflows.

Difficultés rencontrées

Obtenir l'adresse MAC d'un appareil sous Android 11

Comme renseigné dans la documentation d'Android, pour des raisons de sécurité il est impossible de récupérer l'adresse MAC d'un appareil à partir d'Android 11. Ainsi pour contourner le problème, nous générons une adresse MAC aléatoire à l'initialisation de l'appareil.

MqttException - java.net.UnknownHostException

Lors de tests, certains appareils retournent une erreur lors de la tentative de connexion au serveur distant. Nous n'avons pas encore établi la source de cette erreur, ni de façon de la résoudre.

Envoi d'image de taille importante

La fonctionnalité d'envoi d'image est disponible pour des photos de petite taille (screenshots). Dans le cas d'images plus volumineuses, le temps d'envoi est très long, et les messages suivants ne pourront pas être envoyés tant que l'envoi de l'image est en cours.

Affichage sur tablette

Certains aspects de l'application ne sont pas optimisés pour l'affichage sur des écrans de grandes tailles tels que les tablettes par exemple.

Implémentation de tests d’intégration

Ayant peu de pratique et de connaissances sur l’implémentation des tests sur Android, il a été difficile de rédiger des tests d’intégrations complets pour les dynamiques propres à Android. Des tests unitaires plus simples sur les aspects fonctionnels des objets ont donc été réalisés en priorité.

Les tests d’intégrations qui ont été réalisés ont trait aux aspects visuels, et méritent d’être plus approfondis.

Références

[Best practices for unique identifiers | Android Developers]

[Android and MQTT : a simple guide]