**SAE 302 -- Développer des applications communicantes**



**Document de Fonctionnement – Architecture de TraceCord**

**Auteurs : Bastien Labeste, Robin Kwiatkowski, Quentin Chambelland**

**📅 Dernière mise à jour : 16 Février 2025**



**Document d’installation**

Table des matières

[1. Introduction 2](#_Toc190622434)

[2. Schéma global d’architecture 3](#_Toc190622435)

[3. Fonctionnement du Bot Python 4](#_Toc190622436)

[3.1. Récupération des messages – Commande !selection 4](#_Toc190622437)

[3.2. Insertion en base – Commande !java 4](#_Toc190622438)

[4. Base de Données et API PHP 5](#_Toc190622439)

[4.1. Base de Données MySQL 5](#_Toc190622440)

[4.2. Scripts PHP (API) 5](#_Toc190622441)

[5. Application Android 6](#_Toc190622442)

[5.1. Démarrage de l’application 6](#_Toc190622443)

[5.2. Sélection d’un utilisateur → activity\_messages.xml 6](#_Toc190622444)

[5.3. Bouton “Classement” → activity\_classement.xml 6](#_Toc190622445)

[6. Résumé des Étapes de Fonctionnement 6](#_Toc190622446)

[7. Vérifications et Tests 7](#_Toc190622447)

[8. Conclusion 8](#_Toc190622448)

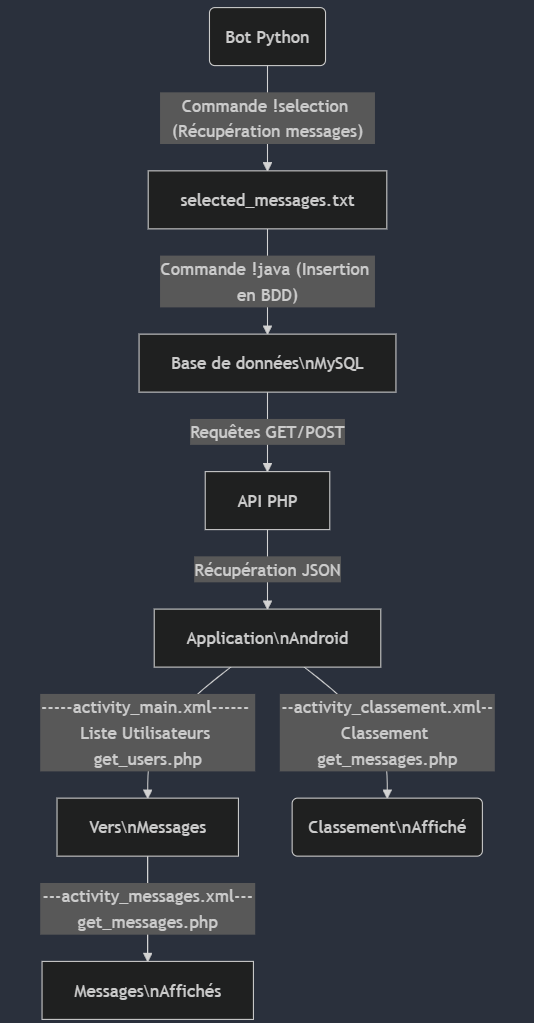
# 1. Introduction

Ce document explique comment les différentes briques du projet TraceCord interagissent entre elles :

* **Le bot Python**, qui récupère les messages depuis Discord ou un terminal.
* **La base de données MySQL**, où sont stockés les messages et les utilisateurs.
* **L’API PHP**, qui fournit les informations aux applications clientes.
* **L’application Android**, qui affiche la liste des utilisateurs et leurs messages.

Vous y trouverez un **schéma des interactions** et les **étapes de fonctionnement** pour comprendre globalement le déroulement.

# 2. Schéma global d’architecture



1. Le **bot Python** exécute une commande !selection (sur Discord ou dans un terminal) pour collecter les messages souhaités, stockés dans le fichier **selected\_messages.txt**.
2. Ensuite, la commande !java (toujours sur Discord ou dans un terminal) permet d’importer ces données dans la **base de données MySQL** à partir de **selected\_messages.txt**.
3. L’**API PHP** (fichiers get\_users.php, get\_messages.php, etc.) permet à toute application cliente d’interagir avec la base de données.
4. Enfin, l’**application Android** interroge l’API :
   * **activity\_main.xml** affiche la liste des utilisateurs via get\_users.php.
   * En sélectionnant un utilisateur, on accède à **activity\_messages.xml**, qui affiche ses messages via get\_messages.php.
   * En cliquant sur le bouton “Classement”, on ouvre **activity\_classement.xml**, qui utilise également get\_messages.php pour afficher le classement de toxicité (ou autre critère).

# 3. Fonctionnement du Bot Python

## 3.1. Récupération des messages – Commande !selection

* Lorsqu’un administrateur ou un utilisateur habilité exécute la commande !selection (sur Discord ou en local via le terminal), le **bot Python** parcourt les messages d’un salon ou d’un certain intervalle.
* Il récupère les informations (nom d’utilisateur, contenu du message, destinataire, date, longueur du message, etc.) et les enregistre dans le fichier **selected\_messages.txt**.

Ce fichier texte sert de passerelle avant l’envoi vers la base de données.

## 3.2. Insertion en base – Commande !java

* Une fois la sélection faite, on exécute la commande !java.
* Cette commande appelle un script (par exemple un fichier Java “DiscordDataProcessor.java” ou un process similaire) qui lit le contenu de **selected\_messages.txt**.
* Les données sont alors insérées dans la base de données **MySQL**.

À ce stade, la base de données **Discord** (ou toute autre nommée) contient de nouveaux messages prêts à être exploités.

# 4. Base de Données et API PHP

## 4.1. Base de Données MySQL

* Les tables principales sont, par exemple, users et messages.
* L’insertion des messages se fait via la commande !java décrite ci-dessus.
* Les champs typiques de la table messages incluent :
  + **id**
  + **nom\_utilisateur**
  + **message**
  + **date**
  + **receveur**
  + **message\_length**
  + **score\_de\_toxicite** (le cas échéant)

## 4.2. Scripts PHP (API)

1. **get\_users.php**
   * Permet de lister tous les utilisateurs stockés dans la table correspondante (ou d’en extraire via la table messages).
   * Retourne un JSON contenant la liste des utilisateurs.
2. **get\_messages.php**
   * Permet de récupérer les messages selon un ou plusieurs paramètres (ex. username=<NOM\_UTILISATEUR>).
   * Retourne un JSON contenant les messages, éventuellement le score de toxicité et d’autres statistiques.
3. **config.php**
   * Fichier de configuration contenant l’hôte, le nom de la base de données, l’utilisateur et le mot de passe.
   * Chargé par les autres fichiers PHP pour se connecter à MySQL.

# 5. Application Android

## 5.1. Démarrage de l’application

* À l’ouverture, on arrive sur l’écran **activity\_main.xml**, qui lance une requête **GET** vers get\_users.php.
* Les utilisateurs reçus en JSON sont affichés dans une liste.

## 5.2. Sélection d’un utilisateur → activity\_messages.xml

1. **Clique sur l’utilisateur** : l’application appelle get\_messages.php avec username=<nom\_utilisateur> en paramètre.
2. **Affichage des messages** :
   * Contenu, date, destinataire, longueur, score de toxicité éventuel, etc.
   * Mise en page personnalisée dans un **RecyclerView** ou **ListView**.

## 5.3. Bouton “Classement” → activity\_classement.xml

1. **Clique sur “Classement”** : redirige vers une nouvelle activité.
2. **Récupération des données** : on peut réutiliser get\_messages.php (ou un autre endpoint) pour analyser les scores de toxicité ou tout autre critère de classement.
3. **Affichage du classement** : par exemple, “du moins toxique au plus toxique” ou selon le nombre de messages envoyés.

# 6. Résumé des Étapes de Fonctionnement

1. **Lancement du bot Python**
   * Commande !selection → **selected\_messages.txt**
   * Commande !java → insertion BDD MySQL
2. **API PHP**
   * Fichiers get\_users.php et get\_messages.php pour fournir des endpoints REST.
3. **Application Android**
   * **MainActivity** (activity\_main.xml) : liste les utilisateurs via get\_users.php.
   * **MessagesActivity** (activity\_messages.xml) : liste les messages par utilisateur via get\_messages.php.
   * **ClassementActivity** (activity\_classement.xml) : classements et statistiques via get\_messages.php.

# 7. Vérifications et Tests

* **Vérifier la bonne exécution du bot Python** :
  + S’assurer que selected\_messages.txt est bien généré après la commande !selection.
  + Contrôler que la commande !java insère correctement les données dans la base (faire un SELECT \* FROM messages;).
* **Vérifier le fonctionnement de l’API** :
  + Tester http://<IP\_SERVEUR>/get\_users.php et http://<IP\_SERVEUR>/get\_messages.php?username=<User> dans un navigateur ou via curl.
  + Vérifier que la réponse en JSON est correcte.
* **Tester l’application Android** :
  + Sur un émulateur ou un appareil physique, voir si la liste des utilisateurs s’affiche.
  + Vérifier si, au clic sur un utilisateur, ses messages apparaissent correctement.
  + Vérifier si le bouton “Classement” affiche la page correspondante avec les bonnes données.

# 8. Conclusion

L’architecture de TraceCord repose sur un **bot Python** pour la collecte initiale des messages, une **base MySQL** pour leur stockage et une **API PHP** pour rendre ces données accessibles. L’**application Android** consomme ces endpoints et propose une interface utilisateur permettant de consulter les utilisateurs, leurs messages et divers classements.

Grâce à cette documentation, vous avez une **vue d’ensemble** de la façon dont chaque composant s’imbrique, depuis l’ajout de données par le bot jusqu’à l’affichage final sur mobile.

**Développé par :**

Bastien Labeste  
Robin Kwiatkowski  
Quentin Chambelland

**Dernière mise à jour :** 16 Février 2025