Ayyub BOUTAHIR
Abasse ALI

L3 IRT

Projet Informatique
Service d'annuaires partagés
Spécification Protocolaire (RFC)

STRI1A/L3 - Informatique
Licence 3 - Informatique, réseaux et télécommunications
Université Paul Sabatier, sciences, ingénierie et technologie

Année 2025 - 2026
Enseignant : André Aoun

SOMMAIRE

1. Contexte et objectifs du projet

Le présent document a pour but de spécifier le protocole de communication d'une application distribuée reposant sur une architecture **client/serveur**. L'objectif principal de ce système est d'offrir **un service de gestion d'annuaires de contacts**, permettant à chaque utilisateur de maintenir ses propres données tout en offrant la possibilité de partager et de consulter les annuaires d'autres utilisateurs au travers d'un réseau.

D'un point de vue technique, l'application se distingue par son mode de stockage. En l'absence de Base de Données Relationnelle (SGBD), le serveur assure la persistance et la centralisation des données exclusivement via l'utilisation de fichiers texte/CSV. Le serveur doit être capable de gérer plusieurs annuaires simultanément et d'associer un annuaire unique à chaque compte utilisateur créé.

Le système définit des rôles distincts pour garantir la sécurité et l'organisation des données :

- **L'Administrateur du serveur** : Il est responsable de la gestion du cycle de vie des comptes utilisateurs (création, modification, suppression). La création d'un compte entraîne automatiquement l'association d'un annuaire à celui-ci.

- **L'Utilisateur** : Une fois authentifié via un menu textuel sécurisé , il dispose des droits pour gérer le contenu de son annuaire (ajout, suppression, modification de contacts contenant nom, prénom, email, téléphone, etc.). Il gère également les permissions d'accès, autorisant ou révoquant la consultation de ses données par d'autres membres.
  +3

Cette RFC (Request For Comments) a pour vocation de définir le format des échanges (PDU), les codes d'erreurs et les scénarios de communication nécessaires pour assurer l'interopérabilité entre le client et le serveur dans ce contexte spécifique.

2. Format des Unités de Données de Protocole (PDU)

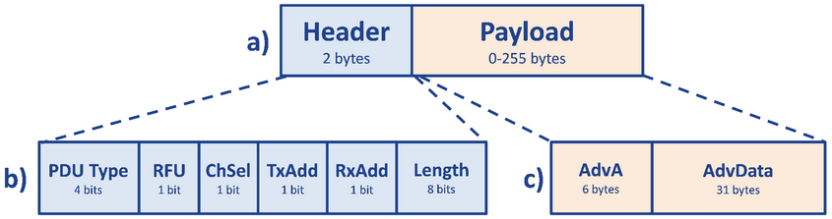
Comme spécifié dans les consignes, les échanges entre le client et le serveur se feront via des messages structurés. Nous avons retenu le format JSON (JavaScript Object Notation) pour sa légèreté et sa facilité d'analyse (parsing).

Chaque PDU sera transmise sous forme de chaîne de caractères encodée en UTF-8.

A. Structure Générale

Une PDU est constituée de deux parties principales :

1. L'En-tête (Champs communs) : Permet d'identifier l'action et la session.
2. Le Corps (Payload) : Contient les données spécifiques à la requête ou à la réponse.



Concrètement, **une PDU (Protocol Data Unit)** agit comme une enveloppe standardisée pour chaque message transitant sur le réseau.

Comme illustré ci-dessus, cette structure binaire ou textuelle permet de séparer clairement deux types d'informations :

- **L'En-tête (Header)** : Il contient les "méta-données" de la communication (ex: le type d'action à effectuer, le jeton d'authentification). C'est ce qui permet au serveur de diriger la requête vers la bonne fonction.
- **Le Corps (Payload)** : Il contient la "charge utile", c'est-à-dire les données concrètes de l'application (ex: le nom et le numéro d'un contact à ajouter).

L'utilisation de PDU définies est indispensable pour garantir l'interopérabilité entre le client et le serveur. Sans ce format strict, le serveur serait incapable d'interpréter correctement le flux de données reçu. De plus, l'encapsulation dans un format JSON (comme spécifié dans l'introduction de cette section) rend ces PDU lisibles par un humain, ce qui facilitera grandement le débogage lors de la phase de développement.

Afin de traduire cette architecture logique en une implémentation technique exploitable, nous avons établi des modèles JSON stricts. Ces modèles formalisent la structure des échanges en appliquant la séparation entre les méta-données de contrôle et les données applicatives.

Les sections suivantes détaillent la syntaxe exacte que doivent respecter les requêtes émises par le client ainsi que les réponses retournées par le serveur :

Format d'une REQUÊTE (Client -> Serveur) :

```
{
  "action": "NOM_DE_L_ACTION",
  "demandeur": "Nom_Utilisateur_Connecté",
  "parametres": {
      // Données spécifiques à l'action
      "parametre_1": "valeur",
      "parametre_2": "valeur"
  }
}
```

Description des champs de la requête :

- action *(Obligatoire)* : Une chaîne de caractères qui sert de "routeur" pour le serveur. Elle indique quelle fonction spécifique doit être déclenchée (ex: `AJOUT_CONTACT`, `CONNEXION`).
- demandeur *(Contextuel)* : L'identifiant de l'utilisateur ou le jeton de session qui garantit l'origine de la demande et permet de vérifier les permissions.
- parametres *(Optionnel)* : Un objet JSON imbriqué contenant les arguments nécessaires à l'action (ex: nom, prénom, email). Ce champ permet de garder la racine du message propre et standardisée.

Format d'une RÉPONSE (Serveur -> Client) :

```
{
  "status": CODE_STATUS,
  "message": "Description lisible du résultat",
  "data": {
      // Données renvoyées (ex: liste de contacts)
  }
}
```

Description des champs de la réponse :

- status *(Obligatoire)* : Un code numérique standardisé (inspiré des codes HTTP comme 200, 404, 500) permettant au client de savoir immédiatement si l'opération a réussi ou échoué.
- message *(Informatif)* : Une explication textuelle du résultat, destinée à être affichée à l'utilisateur ou stockée dans les logs pour le débogage.
- data *(Conditionnel)* : Contient la charge utile demandée par le client (ex: une liste de contacts ou les détails d'un compte). Ce champ reste vide ou absent en cas d'erreur.

---

3. Description des Types de Requêtes et Réponses

Cette section recense les transactions supportées par le protocole, pilotées par le champ action de la PDU qui agit comme routeur pour le serveur. Les opérations sont segmentées en trois domaines : la gestion de session (accès public), l'administration système (rôle Administrateur) et la gestion d'annuaire (rôle Utilisateur). Toute action non répertoriée ici entraînera systématiquement un rejet avec un code d'erreur 400.

A. Gestion de Session (Tout public)

Action	Description	Paramètres Requis (dans corps)	Réponse attendue (donnee)
CONNEXION	Authentifie l'utilisateur en vérifiant ses identifiants dans comptes.csv.	nom, mdp	Rôle de l'utilisateur ("administrateur" ou "utilisateur").

Vous remarquerez l'absence d'une action explicite **DECONNEXION** dans ce protocole. Ce choix architectural est volontaire : la fermeture de session est gérée localement par le client (réinitialisation de son état interne). Le serveur n'a pas besoin de recevoir de PDU spécifique pour clore l'échange ; il cesse simplement de traiter les demandes de cet utilisateur dès que le client arrête de les envoyer.

B. Administration (Rôle Administrateur uniquement)

Ce module regroupe les opérations de maintenance du système, strictement réservées au rôle **Administrateur**. Le serveur vérifie systématiquement les privilèges du demandeur avant d'autoriser la gestion complète des comptes utilisateurs (création, modification, suppression) ainsi que l'accès aux informations globales du serveur.

Action	Description	Paramètres Requis (dans corps)	Réponse attendue (donnee)
CREATION_COMPTE	Crée un nouvel utilisateur et initialise son fichier CSV d'annuaire.	nom, mot_de_passe, statut	Confirmation de création.
SUPPRESSION_COMPTE	Supprime définitivement un compte utilisateur et son annuaire associé.	nom_compte	Confirmation de suppression.
MODIF_COMPTE	Modifie les propriétés d'un compte (mot de passe ou statut).	nom_compte, nouveau_mdp (optionnel), nouveau_statut (optionnel)	Confirmation de modification.
LISTE_COMPTES	Récupère la liste de tous les utilisateurs enregistrés sur le serveur.	*Aucun*	Liste des noms d'utilisateurs.
INFOS_ADMIN	Demande des statistiques ou informations globales sur le serveur.	*Aucun*	Données statistiques (ex: nombre d'inscrits).

Ces opérations assurent la cohérence du stockage : **CREATION_COMPTE** génère automatiquement un fichier CSV dédié, tandis que **SUPPRESSION_COMPTE** le détruit définitivement pour éviter toute donnée orpheline.

C. Gestion d'Annuaire Personnel (Rôle Utilisateur)

Ces actions concernent la modification des données propres à l'utilisateur connecté (le `demandeur`).

Action	Description	Paramètres Requis (dans corps)	Réponse attendue (donnee)
AJOUT_CONTACT	**Ajoute un nouveau contact dans le fichier CSV de l'utilisateur connecté.**	**contact (Objet contenant : Nom, Prenom, Tel, Email, Adresse)**	**Confirmation d'ajout.**
MODIF_CONTACT	**Met à jour les informations d'un contact existant.**	**contact (Objet contenant les nouvelles données et l'identifiant)**	**Confirmation de mise à jour.**

D. Partage et Consultation (Rôle Utilisateur)

Ces actions gèrent les interactions entre les annuaires des différents utilisateurs.

Action	Description	Paramètres Requis (dans corps)	Réponse attendue (donnee)
LISTE_PROPRIO	Liste les annuaires accessibles par l'utilisateur (le sien + ceux partagés avec lui).	*Aucun*	Liste des propriétaires accessibles.
LISTE_CONTACTS	Récupère le contenu de l'annuaire d'un utilisateur cible spécifique.	proprietaire_cible	Liste complète des contacts du cible.
RECHERCHE_CONTACT	Effectue une recherche textuelle dans un annuaire cible.	proprietaire_cible, recherche (mot-clé)	Liste des contacts correspondants.

GERER_PERMISSION	Accorde ou retire le droit de lecture de son annuaire à un tiers.	utilisateur_cible, type ("donner" ou "retirer")	Confirmation de l'opération.
LISTE_DROIT	Affiche la liste des utilisateurs qui ont actuellement le droit de voir mon annuaire.	*Aucun*	Liste des utilisateurs autorisés.

---

4. Codes de Succès et d'Erreurs

Afin de garantir une communication fiable et standardisée, ce protocole adopte un système de codes numériques inspirés du modèle HTTP. Transmis systématiquement dans le champ status de la réponse, ces codes permettent à l'application cliente d'identifier immédiatement la nature du résultat (succès, erreur de saisie ou défaillance technique) et d'adapter son comportement en conséquence.

Code	Type	Signification	Description
200	Succès	OK	L'action a été réalisée avec succès.
201	Succès	Created	La ressource (compte/contact) a été créée.
400	Erreur Client	Bad Request	Format de PDU invalide ou champ obligatoire manquant (ex: email manquant).
401	Erreur Client	Unauthorized	Échec d'authentification ou token invalide.
403	Erreur Client	Forbidden	Droits insuffisants (ex: un Utilisateur tentant une action Admin ou accès annuaire refusé).
404	Erreur Client	Not Found	Utilisateur, annuaire ou contact introuvable.
409	Erreur Client	Conflict	L'élément existe déjà (ex: login déjà pris).
500	Erreur Serveur	Internal Error	Erreur technique côté serveur (ex: échec écriture fichier).

5. Exemples d'Échanges (Scénarios)

Afin de valider la conformité de l'implémentation, cette section détaille les échanges JSON pour plusieurs scénarios types. Chaque exemple présente la paire constituée de la requête émise par le client et de la réponse retournée par le serveur, couvrant à la fois les fonctionnements nominaux (succès) et la gestion des erreurs.

5.1 Authentification (CONNEXION)

Cas 1 : Échec (Mot de passe et/ou nom d'utilisateur incorrect)

Le serveur rejette la demande car le hachage ne correspond pas.

```
// Requête
{
    "action": "CONNEXION",
    "demandeur": null,
    "corps": {
        "nom": "ali",
        "mdp":
"8ab5b85ea2c2f47ac18a8799765d443fb6412164a457b7f5182c08ea2b00c5f38d6a479b70c
0e23f7fae89767566f03269a3fdd48ed0df98c918558dd0af8a8b"
    }
}
```

```
// Réponse
{
    "status": 401,
    "message": "Connexion Échouée"
}
```

Cas 2 : Succès (Administrateur)

Le serveur valide les identifiants et renvoie le rôle de l'utilisateur.

```
// Requête
{
    "action": "CONNEXION",
    "demandeur": null,
    "corps": {
        "nom": "ali",
        "mdp":
"15f8503d4e3ece769a3977ecf1fc3b8a0175a2a05c2ce4ea93d89598a7a0357b987faf80107
954d2c69d246fcd845c8f83f74b5e4d25e737da037dc2dd56bd27"
    }
}
```

```
// Réponse
{
    "status": 200,
    "role": "administrateur"
}
```

5.2 Consultation d'annuaire (LISTE_CONTACTS)

Cas 1 : Échec (Accès interdit)

L'utilisateur ali tente de lire l'annuaire de jean sans permission.

```
// Requête
{
    "action": "LISTE_CONTACTS",
    "demandeur": "ali",
    "corps": {
        "proprietaire_cible": "jean"
    }
}
```

```
// Réponse
{
    "status": 403,
    "message": "Accès refusé"
}
```

Cas 2 : Succès (Consultation de son propre annuaire)

```
// Requête
{
    "action": "LISTE_CONTACTS",
    "demandeur": "ali",
    "corps": {
        "proprietaire_cible": "ali"
    }
}
```

```
// Réponse
{
    "status": 200,
    "donnee": [
        {
            "Nom": "ALI",
            "Prenom": "Abasse",
```

```
            "Telephone": "0744754699",
            "Adresse": "5 Rue Simone Boudet, 31200 Toulouse",
            "Email": "saadannealiabasse@gmail.com"
        },
        {
            "Nom": "BOUTAHIR",
            "Prenom": "Ayyub",
            "Telephone": "0645563125",
            "Adresse": "",
            "Email": "ayyubboutahir@gmail.com"
        },
        {
            "Nom": "JINORO",
            "Prenom": "Gaillor",
            "Telephone": "",
            "Adresse": "5 rue simone boudet, 31200 toulouse",
            "Email": "gjinoro2@gmail.com"
        }
    ]
}
```

5.3 Gestion des Contacts (CRUD)

Ajout d'un contact (Succès)

```
// Requête
{
    "action": "AJOUT_CONTACT",
    "demandeur": "ali",
    "corps": {
        "contact": {
            "Nom": "DANY",
            "Prenom": "Lauriana",
            "Telephone": "",
            "Adresse": "",
            "Email": "dany.lauriana@gmail.com"
        }
    }
}
```

```
// Réponse
{
    "status": 200,
    "message": "Contact ajouté"
}
```

Modification d'un contact (Succès)

Mise à jour du numéro de téléphone.

```
// Requête
{
    "action": "MODIF_CONTACT",
    "demandeur": "ali",
    "corps": {
        "contact": {
            "Nom": "DANY",
            "Prenom": "Lauriana",
            "Telephone": "0745869577",
            "Adresse": "",
            "Email": "dany.lauriana@gmail.com"
        }
    }
}
```

```
// Réponse
{
    "status": 200,
    "message": "Contact mis à jour"
}
```

Suppression d'un contact (Succès)

```
// Requête
{
    "action": "SUPPR_CONTACT",
    "demandeur": "ali",
    "corps": {
        "contact": {
            "Nom": "DANY",
            "Prenom": "Laurianna"
        }
    }
}
```

```
// Réponse
{
    "status": 200,
    "message": "Contact supprimé avec succès"
}
```

5.4 Recherche de Contact (RECHERCHE_CONTACT)

Cas 1 : Échec (Introuvable)

La recherche ne donne aucun résultat.

```
// Requête
{
    "action": "RECHERCHE_CONTACT",
    "demandeur": "ali",
    "corps": {
        "proprietaire_cible": "ali",
        "recherche": "laurianna"
    }
}
```

```
// Réponse
{
    "status": 400,
    "message": "Contact introuvable",
    "donnee": []
}
```

Cas 2 : Succès

Le contact correspondant est renvoyé.

```
// Requête
{
    "action": "RECHERCHE_CONTACT",
    "demandeur": "ali",
    "corps": {
        "proprietaire_cible": "ali",
        "recherche": "lauriana"
    }
}
```

```
// Réponse
{
    "status": 200,
    "donnee": [
        {
            "Nom": "DANY",
            "Prenom": "Lauriana",
            "Telephone": "0745869577",
            "Adresse": "",
            "Email": "dany.lauriana@gmail.com" } ] }
```

5.5 Permissions et Partage (GERER_PERMISSION)

Retrait d'un droit d'accès

```
// Requête
{
    "action": "GERER_PERMISSION",
    "demandeur": "ali",
    "corps": {
        "utilisateur_cible": "ayyub",
        "type": "retirer"
    }
}
```

```
// Réponse
{
    "status": 200,
    "message": "Modification Effectuée"
}
```

Liste des propriétaires accessibles (LISTE_PROPRIO)

Permet de voir avec qui l'utilisateur peut interagir.

```
// Requête
{
    "action": "LISTE_PROPRIO",
    "demandeur": "ali",
    "corps": {}
}
```

```
// Réponse
{
    "status": 200,
    "message": "Succès",
    "donnee": [ "nouman" ]
}
```

5.6 Administration Système

Création de Compte - Échec (Doublon)

Le compte existe déjà.

```
// Requête
{
    "action": "CREATION_COMPTE",
    "demandeur": "ali",
    "corps": {
        "nom": "ali",
        "mot_de_passe": "...",
        "statut": "administrateur"
    }
}
```

```
// Réponse
{
    "status": 409,
    "message": "Le compte 'ali' existe déjà"
}
```

Création de Compte - Succès

```
// Requête
{
    "action": "CREATION_COMPTE",
    "demandeur": "ali",
    "corps": {
        "nom": "nouman",
        "mot_de_passe": "...",
        "statut": "administrateur"
    }
}
```

```
// Réponse
{
    "status": 201,
    "message": "Compte créé avec succès"
}
```

Modification de Compte (MODIF_COMPTE)

```
// Requête
{
    "action": "MODIF_COMPTE",
    "demandeur": "ali",
    "corps": {
        "nom_compte": "chef",
        "nouveau_mdp": "...",
        "nouveau_statut": "administrateur"
    } }
```

```
// Réponse
{
    "status": 200,
    "message": "Compte 'chef' mis à jour avec succès"
}
```

Suppression de Compte (SUPPRESSION_COMPTE)

Cas : Compte inexistant

```
// Requête
{
    "action": "SUPPRESSION_COMPTE",
    "demandeur": "ali",
    "corps": { "nom_compte": "cheff" }
}
```

```
// Réponse
{
    "status": 404,
    "message": "Compte introuvable"
}
```

Cas : Succès

```
// Requête
{
    "action": "SUPPRESSION_COMPTE",
    "demandeur": "ali",
    "corps": { "nom_compte": "chef" }
}
```

```
// Réponse
{
    "status": 200,
    "message": "Compte chef et données supprimés"
}
```

Informations Globales (INFOS_ADMIN)

Retourne la liste complète des utilisateurs et leurs statistiques.

```
// Requête
{
    "action": "INFOS_ADMIN",
    "demandeur": "ali",
    "corps": {}
}
```

```
// Réponse
{
    "status": 200,
    "donnee": [
        {
            "Nom": "ali",
            "Statut": "administrateur",
            "Nb_Contacts": 5,
            "Nb_Annuaires": 0
        },
        {
            "Nom": "ayyub",
            "Statut": "utilisateur",
            "Nb_Contacts": 1,
            "Nb_Annuaires": 0
        },
        // ... (autres utilisateurs)
    ]
}
```

---

6. Diagramme de Séquence des Échanges

Cette section illustre le flux d'interaction entre l'interface utilisateur et la logique métier. Compte tenu de l'architecture retenue pour cette étape du projet, la couche réseau est simulée par des opérations d'écriture et de lecture de fichiers JSON structurés.

Le module intermédiaire `connexion_ClientServeur.py` agit comme une passerelle de transport. Lorsqu'une action est initiée par le client (via `client.py`), ce module encapsule les données dans un PDU (Protocol Data Unit) et les sérialise dans le fichier `pdu_requete.json` situé dans le dossier partagé `donnee_serveur`.

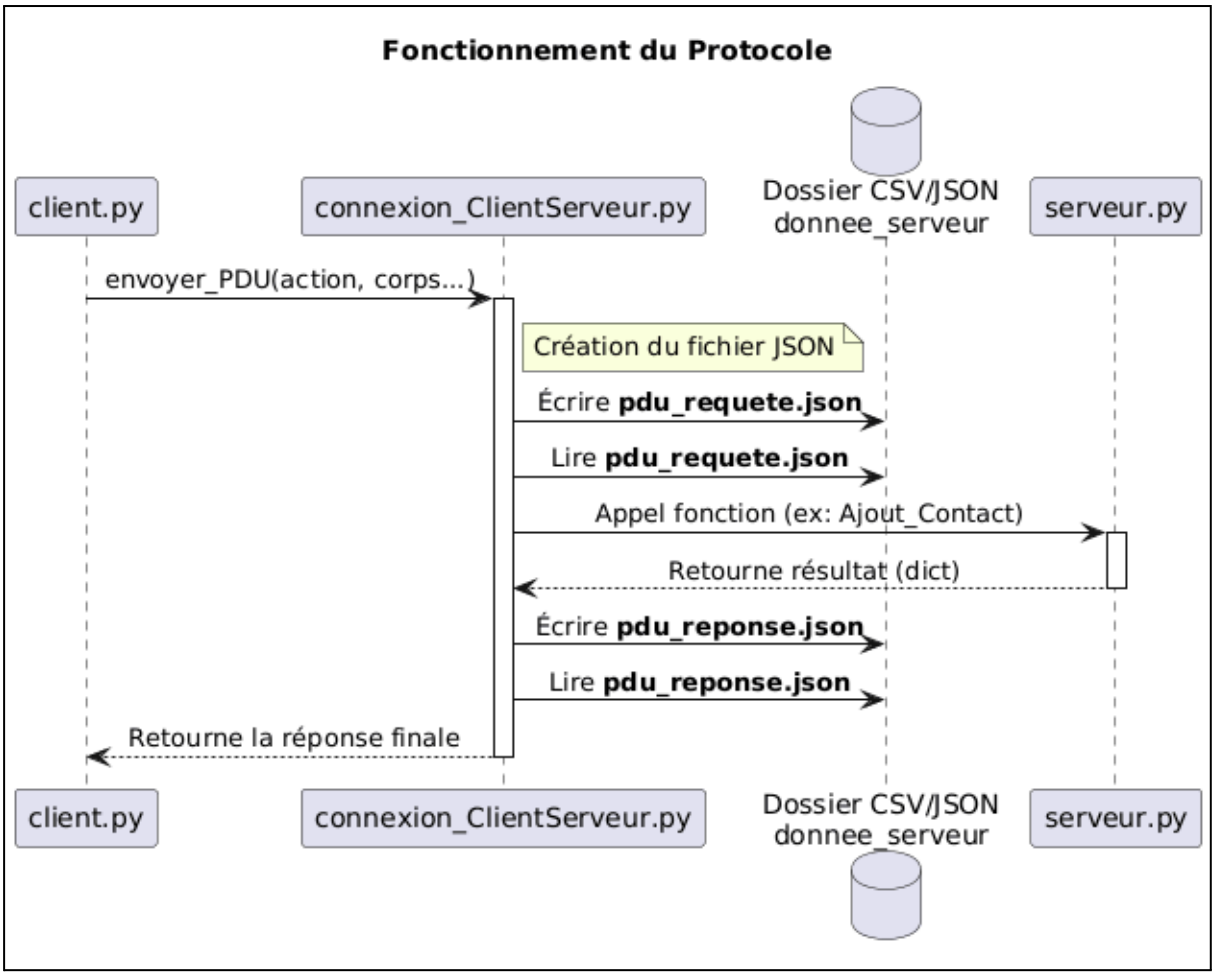


Figure 1 : Flux transactionnel du protocole via échange de fichiers JSON

Le diagramme ci-dessus détaille les étapes d'un cycle complet de requête/réponse :

1. **Émission :** Le client appelle la fonction `envoyer_PDU`.
2. **Transport :** La requête est stockée physiquement sur le disque (`pdu_requete.json`).
3. **Traitement :** Le serveur lit ce fichier, exécute la logique métier via `serveur.py` (ex: vérification de connexion, ajout de contact) et retourne un résultat.
4. **Transport (Retour) :** La réponse est écrite dans `pdu_reponse.json`.
5. **Réception :** Le client récupère les données finales pour affichage.

---

7. Considérations de Sécurité

Cette section décrit les considérations de sécurité relatives à l'utilisation du protocole. Les implémentations du protocole **DOIVENT** prendre en compte les aspects décrits ci-dessous afin de limiter les risques liés à l'authentification, à la confidentialité des données et au contrôle d'accès.

7.1 Authentification et protection des identifiants

Le protocole impose que les informations d'authentification ne soient jamais transmises en clair.

Les mots de passe des utilisateurs **DOIVENT** être transmis sous la forme d'une empreinte cryptographique. L'algorithme de hachage utilisé **DOIT** être une fonction de hachage cryptographique robuste. Cette approche vise à réduire l'exposition des identifiants en cas d'interception des messages.

Le serveur **NE DOIT PAS** stocker ni traiter de mots de passe en clair.

7.2 Contrôle d'accès et permissions

Le protocole repose sur un mécanisme explicite de contrôle d'accès aux annuaires.

Chaque annuaire est associé à un propriétaire unique. Un utilisateur **NE DOIT PAS** accéder à l'annuaire d'un autre utilisateur sans autorisation explicite accordée par le propriétaire de l'annuaire. Le serveur **DOIT** vérifier les permissions associées à chaque requête visant un annuaire tiers. Toute tentative d'accès non autorisé **DOIT** entraîner une réponse avec le code de statut 403 (Interdit).

7.3 Intégrité et cohérence des données

Le serveur **DOIT** assurer la cohérence des données stockées et empêcher les opérations entraînant des conflits ou des incohérences, notamment :

- la création de comptes en double ;
- l'ajout de contacts déjà existants
- la modification ou la suppression de ressources inexistantes.

Les erreurs internes détectées lors du traitement des requêtes **DOIVENT** être signalées au client à l'aide du code de statut 500 (Erreur interne).

7.4 Sécurité du transport

Le protocole ne définit pas de mécanisme natif de chiffrement des échanges.

Lorsque le protocole est déployé sur un réseau non sécurisé, les implémentations **DEVRAIENT** s'appuyer sur une couche de transport sécurisée afin d'assurer la confidentialité et l'intégrité des messages échangés.

---

8. Conclusion générale

Le développement et la spécification du protocole ont permis d'établir un cadre technique rigoureux pour la réponse aux besoins de partage de contacts en réseau. En s'appuyant sur une architecture Client/Serveur éprouvée, ce protocole réussit à concilier simplicité d'implémentation et robustesse opérationnelle.

La force de cette proposition réside dans sa structuration logique qui garantit l'intégrité des échanges de données grâce à quatre piliers fondamentaux :

- Une standardisation des échanges : L'adoption d'un format de messages explicitement défini et l'utilisation de codes de statut normalisés assurent une prévisibilité totale des interactions, facilitant ainsi le débogage et la maintenance côté client comme côté serveur.
- Une grammaire d'actions claire : La délimitation d'un ensemble fini et cohérent d'actions permet de couvrir l'intégralité du cycle de vie des annuaires sans introduire de complexité superflue.
- Une sécurité granulaire : L'intégration native d'un mécanisme de contrôle d'accès fondé sur des permissions explicites offre une gestion fine des droits utilisateurs, élément critique pour la confidentialité des données partagées.
- Une approche "Design-for-Simplicity" : La volonté de conserver un protocole léger favorise une adoption rapide par les développeurs et limite la dette technique potentielle.

8.1 Perspectives et Évolutions

Bien que la version actuelle fournisse une base cohérente et fonctionnelle, son architecture a été pensée pour l'extensibilité.

Le protocole est prêt à accueillir des couches supérieures sans remettre en cause son fondement :

8.2 Évolutions futures envisageables :

- Intégration de mécanismes avancés de sécurité du transport (tels que TLS/SSL).
- Mise en place d'une journalisation centralisée pour l'audit.
- Gestion optimisée des sessions persistantes pour réduire la charge réseau.

En somme, le protocole constitue une fondation solide, capable de supporter la mise en œuvre de services de gestion d’annuaires partagés performants, tout en restant ouvert aux adaptations futures nécessaires à un environnement technologique en constante mutation.