Guide d'Intégration API Auth Service

TraMaSys Engineering

2 décembre 2025

Table des matières

1 Introduction

Ce document détaille les procédures d'intégration avec le microservice d'authentification **TraMaSys Auth**. Ce service gère l'identité des utilisateurs, la sécurité via JWT et le contrôle d'accès basé sur les rôles (RBAC).

L'API suit les standards REST et expose une documentation interactive Swagger à l'adresse :

`https://auth-service.pynfi.com/swagger-ui/index.html#/`

2 Authentification Sécurité

L'API est entièrement *Stateless*. Aucune session n'est conservée côté serveur. Chaque requête vers une route protégée doit inclure un jeton d'accès (Access Token).

2.1 Obtenir un Token (Login)

Pour commencer, l'utilisateur doit s'authentifier avec ses identifiants (Username/E-mail et Mot de passe).

Requête HTTP

POST `/api/auth/login`

L'attribut ***'identifier'*** peut être soit le ***username***, ***l'email*** ou le ***numéro de téléphone***

Listing 1 – Exemple de Body de requête

```
{
  "identifier": "brayan_dev",
  "password": "MonSuperMotDePasse"
}
```

Listing 2 – Exemple de Réponse (Succès 200)

```
{
  "accessToken": "eyJhbGciOiJIUzI1NiJ9...",
  "refreshToken": "eyJhbGciOiJIUzI1NiJ9...",
  "user": {
    "id": "a1b2c3d4-...",
    "username": "brayan_dev",
    "roles": ["ADMIN"],
    "permissions": ["auth:user:delete", "stock:item:read"]
  }
}
```

2.2 Utiliser le Token (Bearer Auth)

Une fois l'`accessToken` obtenu, il doit être envoyé dans les en-têtes (Headers) de **toutes** les requêtes subséquentes.

Attention

Le format de l'en-tête doit respecter scrupuleusement la syntaxe suivante :

— **Header Name** : `Authorization`
— **Header Value** : `Bearer <votreToken>`
Exemple d'en-tête HTTP brut :

```
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY...
```

Si ce header est manquant ou mal formé, l'API retournera une erreur **401 Unauthorized**.

3 Gestion des Droits (RBAC)

Le système repose sur un contrôle d'accès basé sur les rôles (RBAC). Cependant, la granularité fine est gérée par des **Permissions**.

Un utilisateur possède des Rôles (ex : `MANAGER`), et ces Rôles possèdent des Permissions (ex : `stock:produit:create`).

3.1 Nomenclature des Permissions

Pour garantir une cohérence entre tous les services de l'écosystème TraMaSys, les permissions **doivent** respecter une convention de nommage stricte.

Format Obligatoire

`[service] : [Objet] : [action]`

- **[service]** : Nom de l'application ou du microservice concerné (ex : `auth`, `syndicat`, `rideAndGo`).
- **[Objet]** : La ressource manipulée (ex : `user`, `product`, `invoice`).
- **[action]** : L'action effectuée (ex : `create`, `read`, `update`, `delete`, `approve`).

Exemples valides :

- `auth:user:delete` (Droit de supprimer un utilisateur dans le service Auth)
- `stock:inventory:read` (Droit de voir l'inventaire dans le service Stock)
- `rh:contract:sign` (Droit de signer un contrat dans le service RH)

3.2 Cycle de Vie et Dépendances

L'attribution des droits suit une séquence logique stricte. Vous ne pouvez pas lier des éléments qui n'existent pas encore en base de données.

1. **Création de la Permission** : Définir l'action atomique (respectant la nomenclature).
2. **Création du Rôle** : Définir le profil métier (ex : `ADMIN`, `DRIVER`).
3. **Association** : Lier la Permission au Rôle.
4. **Affectation** : Donner le Rôle à un Utilisateur.

3.2.1 Étape 1 : Créer une Permission

POST /api/permissions

Il est impératif de créer la permission avant de pouvoir l'utiliser.

```
{
  "name": "rideAndGo:vehicule:create",
  "description": "Permet de creer une nouveau vehicule"
}
```

3.2.2 Étape 2 : Créer un Rôle

POST /api/roles

Créez le conteneur logique qui regroupera plusieurs permissions.

```
{
  "name": "MAGASINIER"
}
```

3.2.3 Étape 3 : Assigner la Permission au Rôle

POST /api/permissions/assign/{roleName}

Lie la permission créée à l'étape 1 au rôle créé à l'étape 2.

```
// POST /api/permissions/assign/MAGASINIER
{
  "permissionName": "rideAndGo:vehicule:create"
}
```

4 Gestion des Utilisateurs

Une fois les Rôles et Permissions configurés, l'étape finale consiste à gérer les utilisateurs et leurs accès.

4.1 Attribution de Rôles

L'utilisateur n'hérite pas directement de permissions. Il hérite de **Rôles**, qui eux contiennent les permissions. C'est ce mécanisme qui permet de charger automatiquement les droits dans l'attribut `permissions` de l'objet User lors de la connexion.

POST /api/users/{id}/roles/{roleName}

Ajoute un rôle existant à un utilisateur.

Exemple : Promouvoir l'utilisateur "brayan" (ID 123...) en ADMIN.
Paramètres d'URL :
— `id` : UUID de l'utilisateur.
— `roleName` : Nom du rôle (ex : `ADMIN`).
Aucun corps (Body) n'est requis pour cette requête.

DELETE /api/users/{id}/roles/{roleName}

Retire un rôle à un utilisateur (Révocation d'accès).

4.2 Mise à jour du Profil

Permet de modifier les informations personnelles sans toucher aux données sensibles (mot de passe, rôles).

PUT /api/users/{id}

Mise à jour des informations de base.

```
{
  "firstName": "Brayan",
  "lastName": "Armel",
  "phone": "+237600000000"
}
```

4.3 Sécurité : Changement de Mot de Passe

Pour des raisons de sécurité, le changement de mot de passe nécessite de fournir l'ancien mot de passe pour validation.

PUT /api/users/{id}/password

Route sécurisée pour la rotation de mot de passe.

```
{
  "currentPassword": "AncienPassword123!",
  "newPassword": "NouveauPasswordSecure456!"
}
```

5 Conclusion

Ce service d'authentification est conçu pour être la pierre angulaire de la sécurité de l'écosystème TraMaSys.

Rappel des bonnes pratiques :

1. Ne jamais stocker les mots de passe en clair (le service les hache automatiquement).
2. Toujours passer par l'étape de création de permission (`[service]:[objet]:[action]`) avant de tenter une assignation.
3. Utiliser le `refreshToken` pour renouveler les sessions sans demander à l'utilisateur de se reconnecter.

Pour toute assistance technique, veuillez vous référer à l'équipe Backend.