Accès API

Fonctionnement & exemple

Cette documentation permet de mieux comprendre le fonctionnement d'accès API en s'appuyant sur l'exemple de la consigne Dynamique/Manuel de smartCharging.

Structure

La table **public_api_clients** permet de générer une clé API relié à un rôle, elle contient alors :

- `id`
- `roleUserGroupId` : pointe vers `id` de **role_users_groups**
- `entityName` : nom du client
- `clientCode` : nom **unique** équivalent slug
- `apiKeyHash` : clé api hashée
- `features` : fonctionnalités autorisées par la clé
- `status` : état de la clé

Seul le rôle **MANAGER** peut gérer les clés des autres utilisateurs. Voici les routes principales :

- **POST** `/public-api-clients/me` : pour générer une clé sur son rôle
- **POST** `/users/:idOrUuid/role/:roleId` : pour générer la clé d'un id_role_users_groups
- **GET** `/public-api-clients/me` : pour récupérer les informations concernant ma clé
- **PATCH** `/public-api-clients/:id/features` : pour définir les fonctionnalités autorisés par la clé (vierge à la génération du POST)
- **POST** `/public-api-clients/:id/regenerate-key` : pour regénérer la clé
- **POST** `/public-api-clients/:id/revoke` : pour révoquer l'état d'une clé

Un Guard **CombinedAuthGuard** a été mis en place pour permettre d'utiliser une route soit via **jwtToken** (Supervision) ou par **basic token** (clé API). Le Guard **ApiScopeGuard** permet quant à lui d'assurer des droits autorisés pour la clé.

Exemple concret consigne smartCharging externe

Dans notre cas, on souhaite autoriser un client externe à pouvoir modifier la puissance d'une station. Pour se faire, on créé un utilisateur MANAGER ou TECHNICIAN sur le groupe désiré, afin qu'il ait les droits de modifier une station. Pour autant, on ne communiquera pas le mot de passe du compte puisque c'est la clé API générée qui sera utilisée uniquement. Ainsi, grâce à la mise en place de **ApiScopeGuard** et de **features**, on évite que le client puisse interférer sur d'autres routes.

```
  @UseGuards(CombinedAuthGuard, AuthorizationGuard, ApiScopeGuard)
  @ROLES(Role.MANAGER, Role.TECHNICIAN)
  @ApiScope("locations:update")
  @ApiParamIdOrSlug
  @Put(":idOrSlug")
async update(
    @ParseIdOrSlug() idOrSlug: LocationParamIdOrSlugDto,
    @Body() location: LocationUpdateDto,
    @SessionGroups() inGroups: number[],
    @Req() req: Record<string, unknown>,
  ): Promise<number> {}
```

L'utilisation de **CombinedAuthGuard** permet également aux utilisateurs connectés en supervision de pouvoir toujours utiliser la route via **jwtToken**.

Revision #3
Created 6 May 2026 09:11:00 by Clement OLIVIER
Updated 6 May 2026 09:36:00 by Clement OLIVIER