Documentation API Primmo (HApi) pour intégrateur

Cette documentation s'adresse aux intégrateurs techniques et a pour objectif de guider le développement d'interface entre l'API Primmo et divers systèmes tiers, tels que des CRM (Customer Relationship Management), des portails destinés aux locataires, des systèmes de paiement automatisé, des sites web maison, et d'autres solutions métiers. Elle fournit des instructions détaillées sur la manière de configurer et d'établir des connexions fiables et sécurisées entre Primmo et ces plateformes. Grâce à cette intégration, les systèmes tiers pourront échanger des données en temps réel avec l'API, automatisant ainsi des processus clés comme la gestion des baux, la gestion des comptes à recevoir, ou la communication entre les locataires et les administrateurs ou utilisateurs. Ces exemples ne représentent qu'une infime partie des multiples possibilités qui s'offre à l'intégrateur afin de connecter Primmo à des systèmes tiers.

L'API Primmo utilise plusieurs outils et technologies, tels qu'OData, Swagger, Postman, JWT ainsi que les webhooks. Leur utilisation facilite non seulement l'intégration des systèmes tiers, mais améliore également les performances globales de l'interface. Grâce à ces outils, les développeurs peuvent interagir plus efficacement avec l'API, simplifier les tests, assurer une gestion sécurisée des authentifications, et mettre en place des notifications en temps réel. Cela garantit une intégration fluide et performante, réduisant ainsi les complexités techniques tout en optimisant les échanges de données entre les systèmes.

Voici une brève présentation de chaque technologie et de ses avantages.

OData

OData (Open Data Protocol) est un protocole standardisé qui facilite l'échange de données via des services web en utilisant des technologies comme HTTP, AtomPub et JSON. Basé sur le modèle REST, il permet de créer, lire, mettre à jour et supprimer des données de manière cohérente. OData permet d'interroger des ressources via des requêtes URI pour filtrer, trier ou paginer des données. Il est largement utilisé pour l'intégration de systèmes hétérogènes, grâce à son support des métadonnées, qui aide les clients à comprendre et interagir avec les structures de données fournies.

Veuillez consulter la section OData en annexe pour des exemples d'utilisation.

Swagger

Swagger est un outil essentiel pour documenter, concevoir et tester des API REST. Il permet aux développeurs de définir de manière claire les points d'accès, les paramètres, et les réponses d'une API dans un format lisible par l'homme et la machine selon la spécification OpenAPI. Swagger facilite la collaboration entre équipes, car il génère automatiquement une documentation interactive, où les utilisateurs peuvent tester les points d'accès en direct. Il garantit également la cohérence entre la spécification et l'implémentation de l'API, réduisant les

erreurs. En résumé, Swagger améliore la transparence, la maintenabilité, et l'accessibilité des API tout au long de leur cycle de vie.

La documentation Swagger pour l'environnement *staging* est disponible à l'adresse ci-dessous

https://primmoapistaging.hopemservices.com/swagger/index.html

Collection Postman

Utiliser une collection Postman offre plusieurs avantages pour gérer et tester des API de manière structurée. Une collection regroupe plusieurs requêtes API organisées en dossiers, facilitant leur gestion et exécution par étapes. Cela permet de tester différents scénarios et flux de travail de manière cohérente. De plus, Postman permet de sauvegarder des environnements (comme développement, test, *staging*, production), d'automatiser les tests avec des scripts, et de partager facilement la collection avec d'autres membres d'une équipe. Elle aide également à vérifier les réponses, gérer les authentifications et monitorer les performances, rendant le processus de développement et de test des API plus efficace et collaboratif.

La collection est disponible sur demande pour les intégrateurs.

Documentation API Primmo

La description détaillée de chaque entité est accessible dans la documentation complète de l'API Primmo. Celle-ci fournit des informations exhaustives sur la structure, les attributs et les relations des entités, permettant aux intégrateurs de comprendre comment manipuler et interagir efficacement avec les données de l'API. Cette documentation sert de référence pour toutes les entités exposées par l'API, facilitant ainsi l'intégration des fonctionnalités dans des systèmes tiers. Les développeurs peuvent y consulter les spécificités techniques nécessaires pour réaliser des appels API précis et optimiser les interactions avec les différents services offerts par Primmo.

Webhooks

Un w**ebhook** est un mécanisme important qui permet à une application d'envoyer des notifications ou des données à une autre application en temps réel via des requêtes HTTP. Il est souvent utilisé pour faciliter la communication entre différents systèmes sans nécessiter de vérifications constantes (comme un *polling*). L'API Primmo met à disposition des webhooks sur les entités les plus utilisées. Veuillez consulter la section webhooks en annexe pour une description complète de leur utilité dans une API.

Comment utiliser les fonctions de l'API

Cette section décrit comment faire un premier appel à l'API. Il est recommandé d'utiliser l'outil Postman ou un outil similaire afin de tester la connectivité. Dans l'exemple ci-dessous, les étapes sont décrites afin d'obtenir la liste des immeubles.

Authentification

1- La première étape consiste à obtenir un jeton (token) d'authentification

L'URL est **{url}/api/Token**

Le corps du message (body), au format json, doit contenir les informations de l'utilisateur ainsi que son mot de passe :

```
POST {{baseUrl}} /api/Token
{
    "UserName":"myUsername",
    "Password":"myPassword"
}
```

La clé API qui vous a été fournie doit être passée comme valeur dans les *Headers* avec la clé **x-api-key**

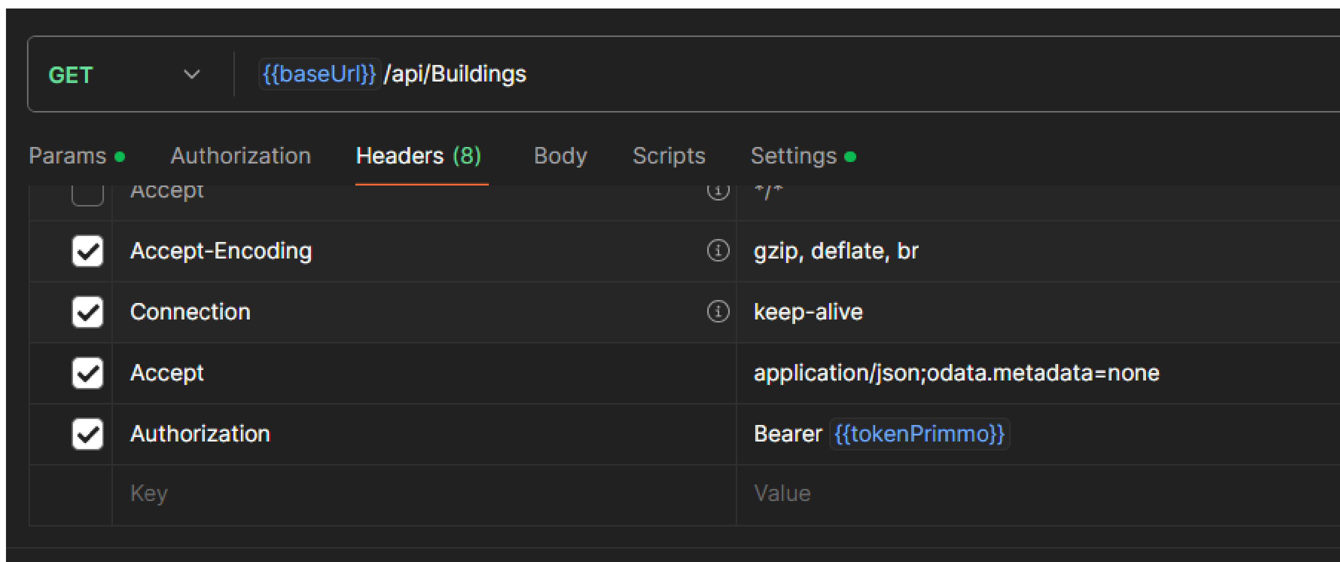
Key	Value
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.42.0
Accept	*/*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
x-api-key	615AA377-442E-8EAA-AD7B-D38AFBD6BC74

Le *token* sera retourné dans le corps du message, et réutilisé à la prochaine étape.

2- Obtenir la liste des immeubles (par exemple)

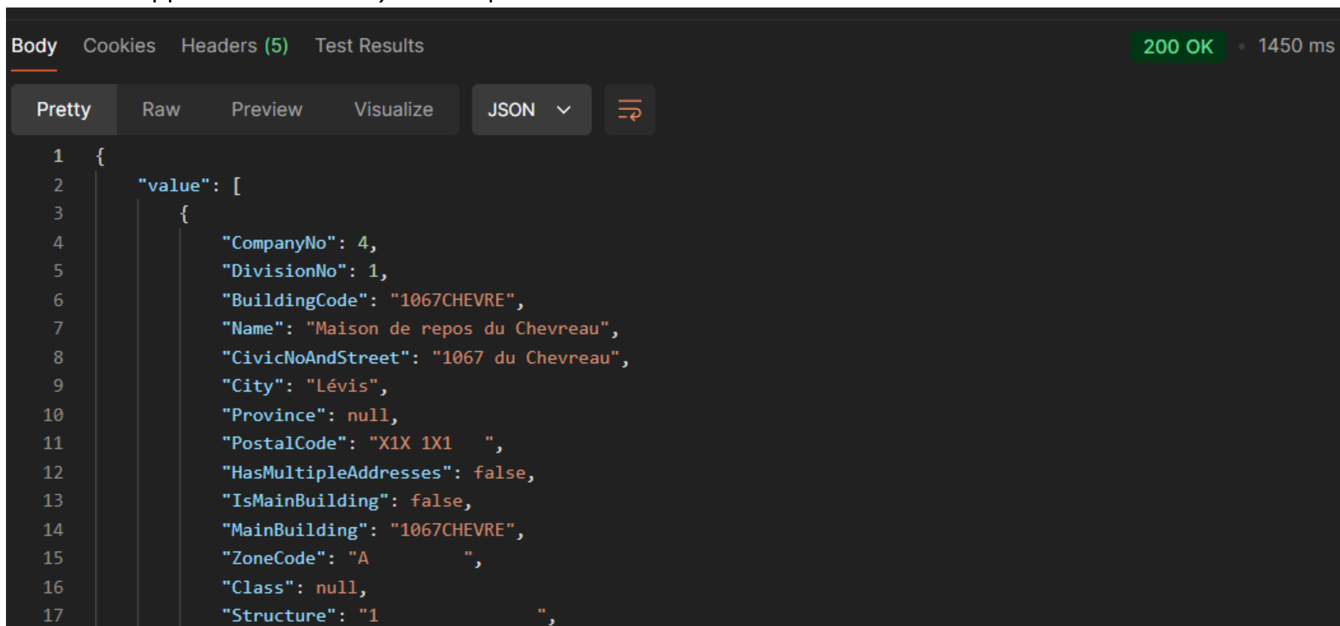
L'URL est **{url}/api/Buildings**

Le *token* (Bearer) doit être passé dans les *Headers* avec la clé **Authorization.** La valeur doit être préfixée de "Bearer "



Ici {{tokenPrimmo}} prend la valeur récupérée à l'étape 1

Le résultat apparait dans le *body* de la réponse :




```
{
    "value": [
        {
            "CompanyNo": 4,
            "DivisionNo": 1,
            "BuildingCode": "1067CHEVRE",
            "Name": "Maison de repos du Chevreau",
            "CivicNoAndStreet": "1067 du Chevreau",
            "City": "Lévis",
            "Province": null,
            "PostalCode": "X1X 1X1   ",
            "HasMultipleAddresses": false,
            "IsMainBuilding": false,
            "MainBuilding": "1067CHEVRE",
            "ZoneCode": "A         ",
            "Class": null,
            "Structure": "1               ",
```

Puisque le *token* expire après 24 heures, il est nécessaire d'en regénérer un nouveau après son expiration. Lors de l'utilisation de Postman ou de tout autre outil, il est recommandé de configurer des variables pour gérer automatiquement les valeurs de **x-api-key** et du *token*. Cela simplifie le processus de gestion des authentifications, car les variables permettent de réutiliser ces informations dans plusieurs requêtes sans avoir à les mettre à jour manuellement à chaque fois. Cette approche garantit une meilleure flexibilité et efficacité dans les tests et appels API, tout en réduisant les erreurs liées à la manipulation des clés et des *tokens* d'authentification.

Cela devrait être suffisant pour être en mesure de faire les premiers appels à l'API.

Afin d'optimiser les requêtes à l'API et de réduire de façon significative le temps de réponse, il peut être avantageux d'utiliser les fonctionnalités OData. Ces fonctionnalités sont décrites à la section suivante.

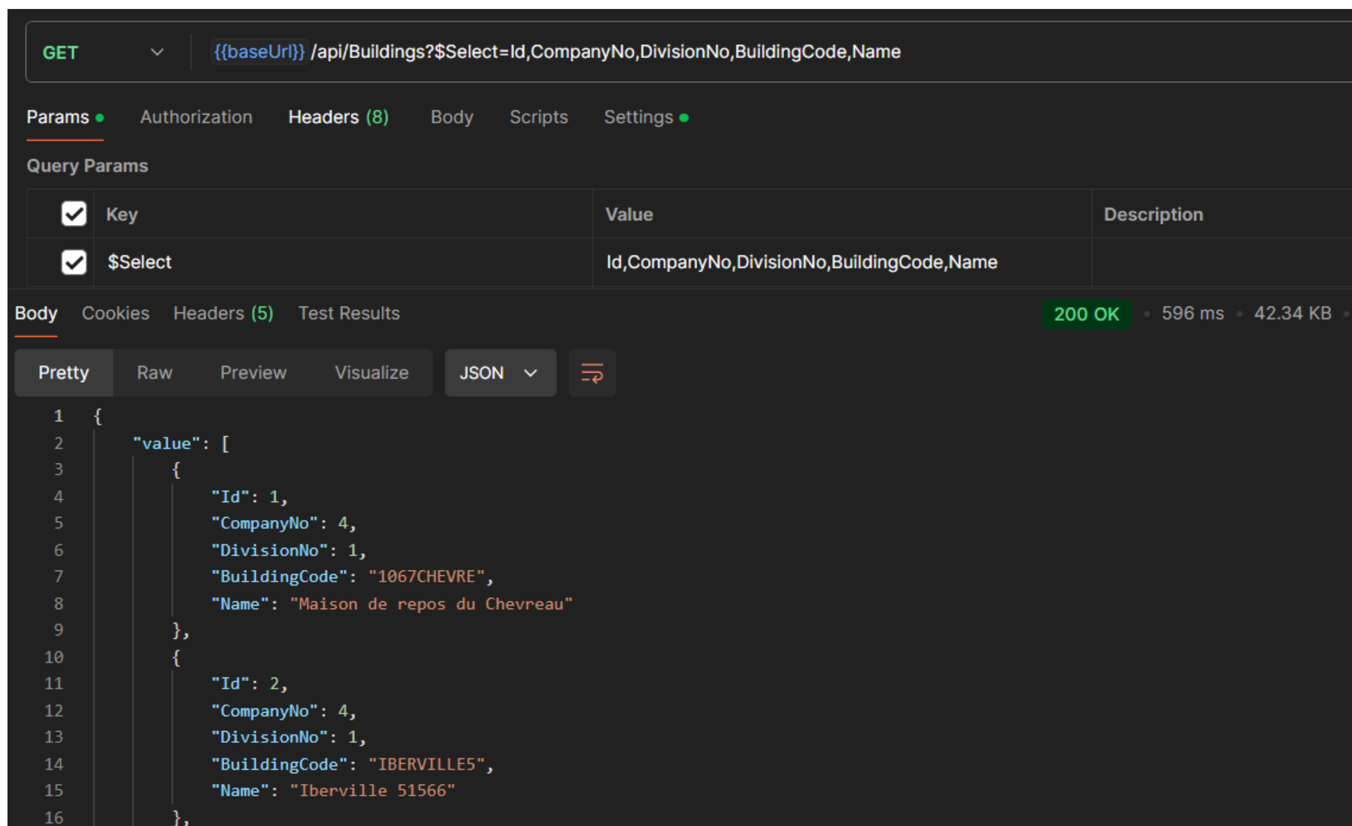
OData

OData propose une variété de clauses permettant d'optimiser les requêtes de données. En offrant des options telles que les filtres, les tris, et la sélection de champs spécifiques, OData facilite l'accès ciblé et efficace aux informations, réduisant ainsi la charge sur les systèmes et améliorant les performances des applications. Voici des exemples de clauses OData

$select

La clause $select dans OData permet de spécifier les champs ou propriétés d'une entité que l'on souhaite récupérer dans une requête, réduisant ainsi la quantité de données renvoyées. Elle optimise les requêtes en ne renvoyant que les informations nécessaires, améliorant les performances et l'efficacité du traitement des données.

Ex : Comment obtenir la liste de tous les immeubles, mais seulement les informations suivantes sont requises : l'identifiant unique, le numéro de compagnie, le numéro de division, le code et le nom de l'immeuble.



$filter

La clause $filter dans OData permet de restreindre les résultats d'une requête en spécifiant des conditions sur les propriétés des entités. Cela permet de récupérer uniquement les données qui répondent à des critères spécifiques, améliorant ainsi la pertinence des résultats et l'efficacité des requêtes.

Ex : Obtenir la liste de toutes les unités de la compagnie 3, division 1, pour l'immeuble 1000PINS.

GET {{baseUrl}} /api/Units?$filter=CompanyNo eq 3 and DivisionNo eq 1 and BuildingCode eq '1000PINS'

Params Authorization Headers (8) Body Scripts Settings

Key	Value	Description
$filter	CompanyNo eq 3 and DivisionNo eq 1 and BuildingCode eq '1000PINS'	

Body Cookies Headers (6) Test Results 200 OK 592 ms 222.75 KB

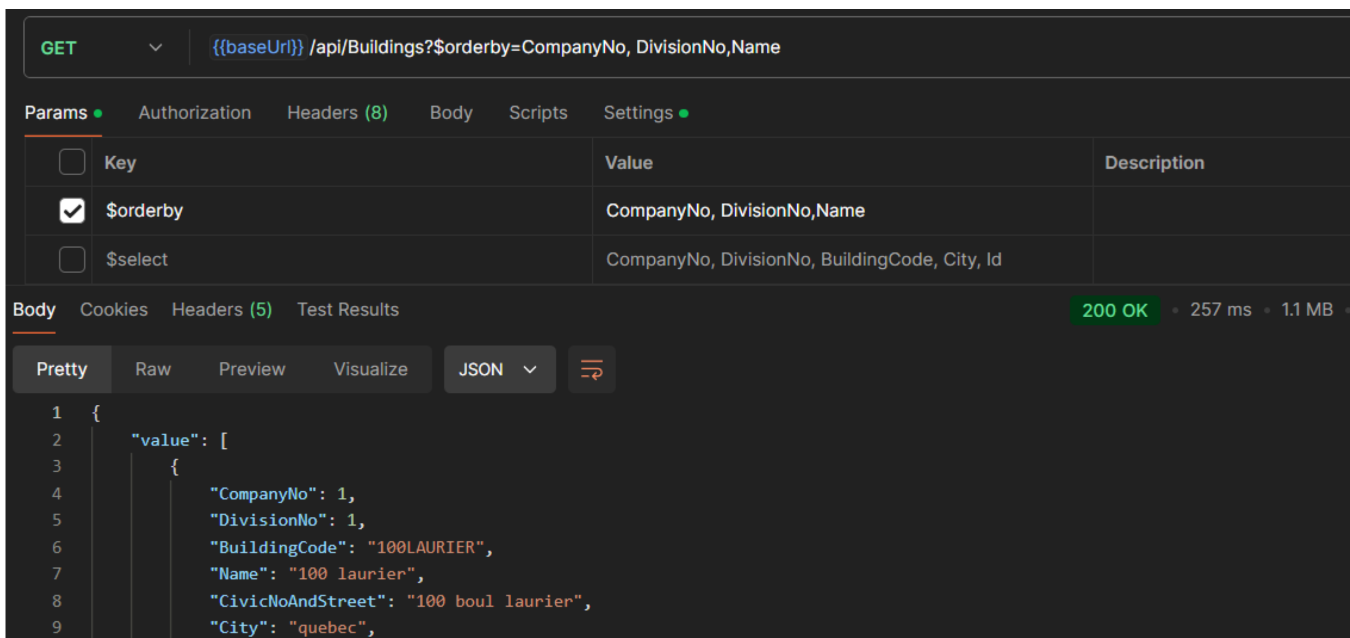
Pretty Raw Preview Visualize JSON

```
1  {
2      "value": [
3          {
4              "CompanyNo": 3,
5              "DivisionNo": 1,
6              "BuildingCode": "1000PINS  ",
7              "UnitCode": "       ",
8              "OccupancyStatus": "V ",
9              "HousingType": null,
10             "NumberOfRooms": 4.5,
```

$orderby

La clause $orderby dans OData permet de trier les résultats d'une requête selon une ou plusieurs propriétés des entités. Cela facilite l'organisation des données récupérées, permettant aux utilisateurs de visualiser les résultats dans un ordre spécifique, comme croissant ou décroissant, et d'améliorer l'expérience de navigation et d'analyse des données.

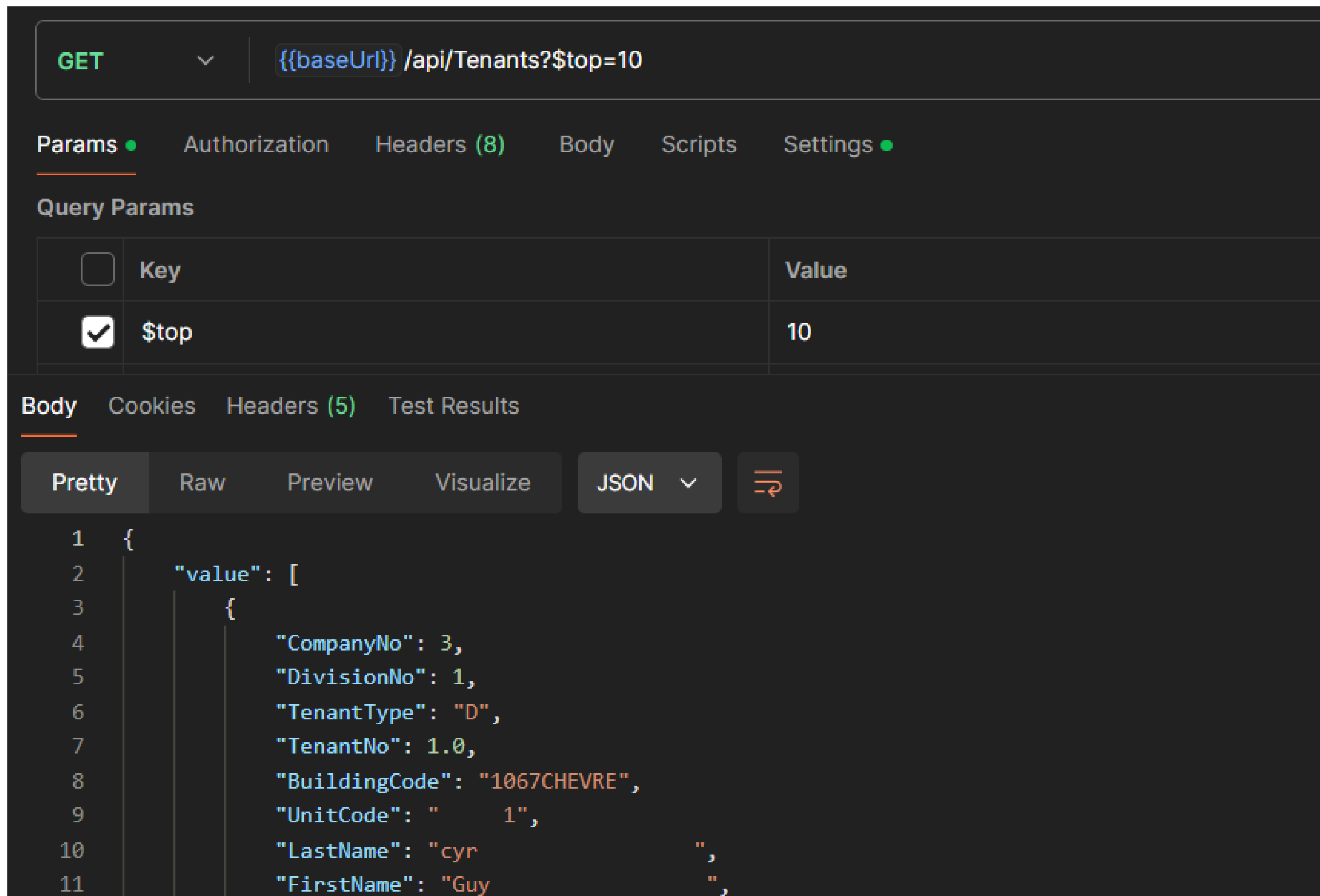
Ex : Obtenir la liste des immeubles par ordre de compagnie, division et nom de l'immeuble.



$top et $skip

La clause $top dans OData permet de limiter le nombre de résultats renvoyés par une requête en spécifiant un nombre maximal d'entrées à retourner. En parallèle, la clause $skip permet d'ignorer un nombre défini d'enregistrements dans les résultats, facilitant ainsi la pagination des données. Ensemble, ces clauses permettent de gérer efficacement de grands ensembles de données, offrant aux utilisateurs la possibilité d'accéder à des portions spécifiques tout en améliorant la performance des requêtes.

Ex : Requête permettant de retourner les 10 premiers locataires

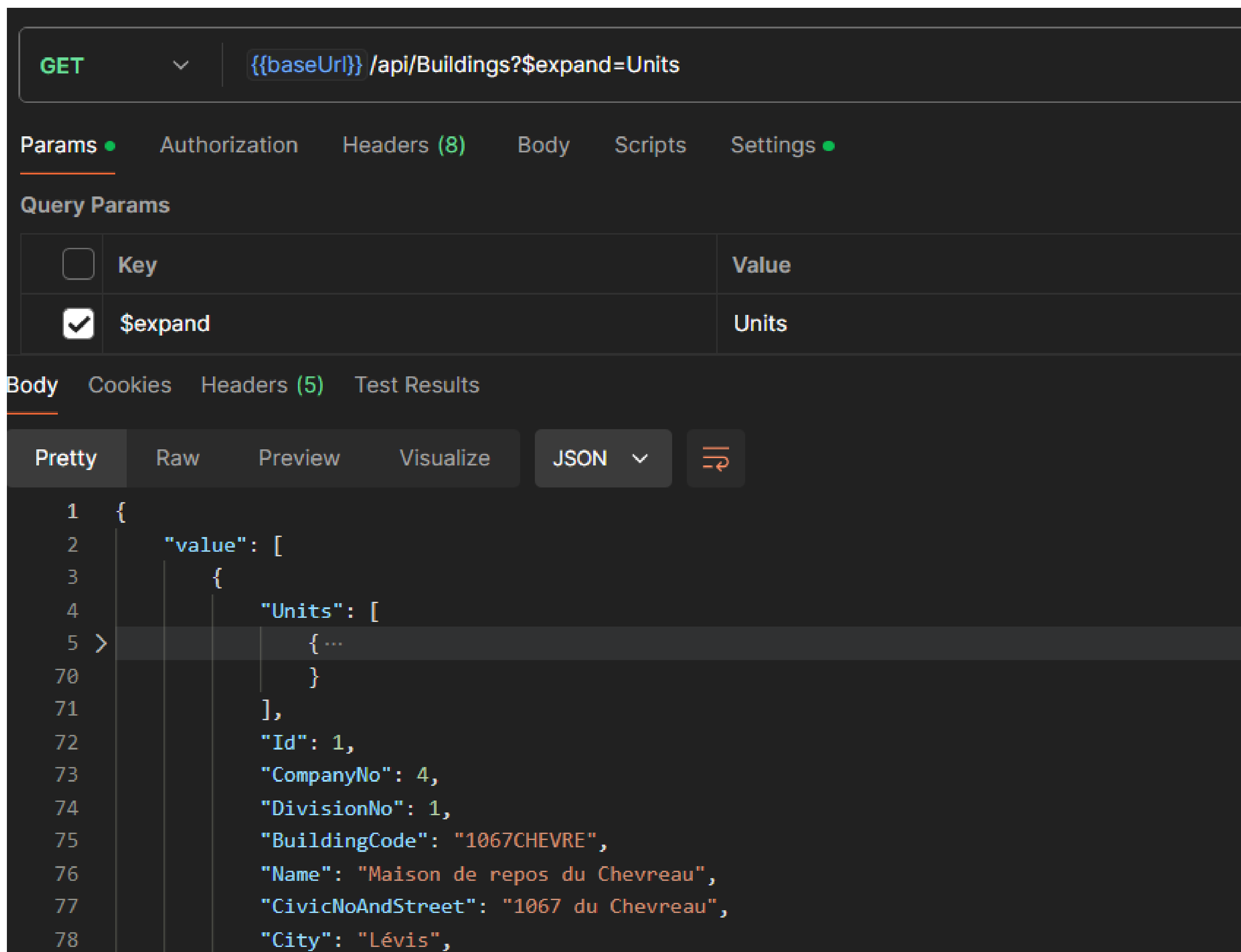


Ces 2 clauses combinées peuvent être très utiles pour un système navigant au travers un volume de données important nécessitant une pagination.

$expand

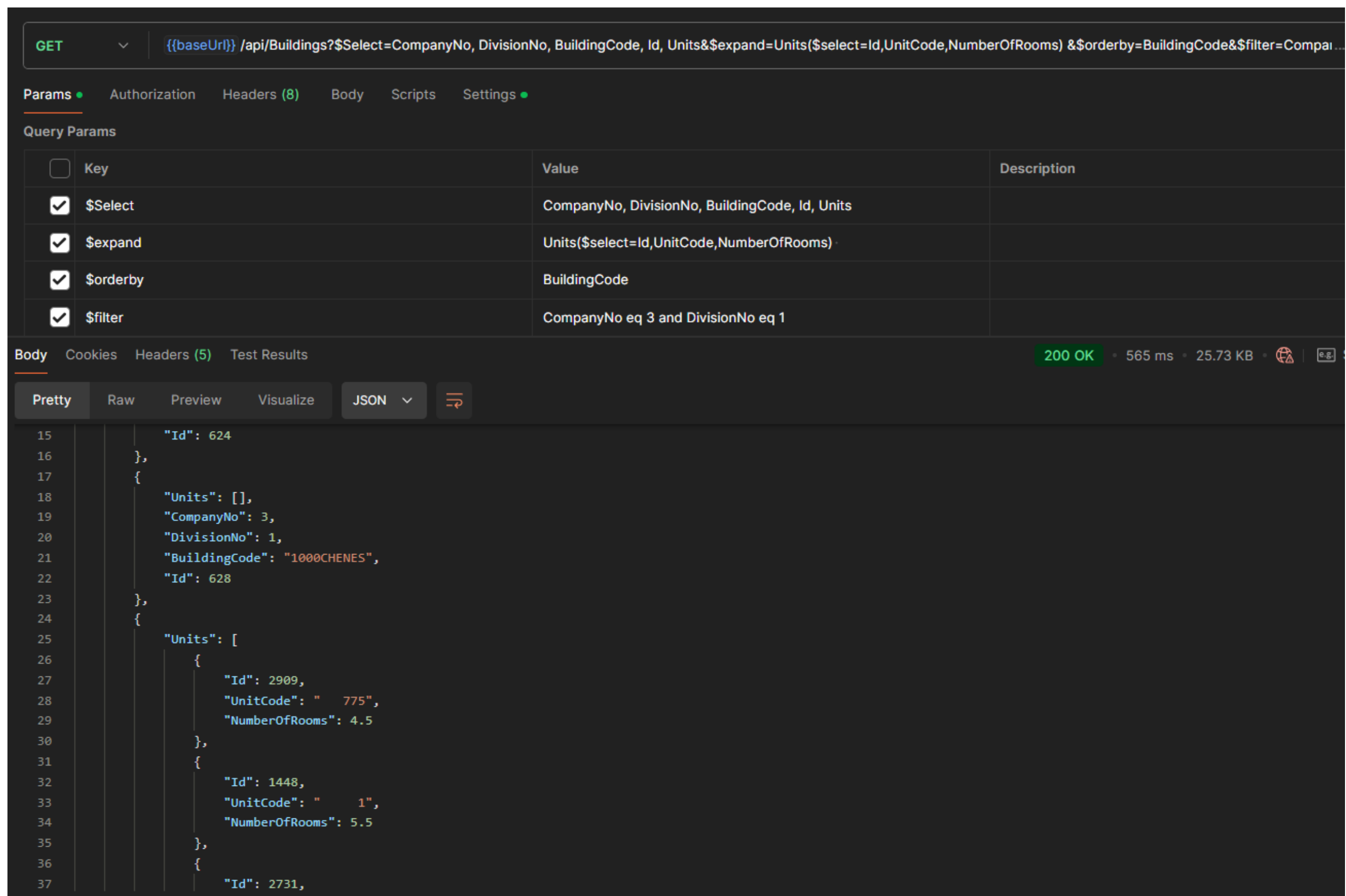
La clause $expand dans OData permet de récupérer des entités associées en une seule requête en incluant des données liées dans les résultats. Cela simplifie les appels API en évitant des requêtes supplémentaires pour obtenir des relations, ce qui améliore l'efficacité des échanges de données et facilite la navigation dans les structures d'entités liées, comme les collections ou les sous-entités.

Ex : Requête permettant de retourner la liste des unités liées à chacun des immeubles



Notez que ces clauses peuvent être utilisées en combinaison afin d'effectuer des requêtes précises et optimales.

Ex : Obtenir la liste des immeubles de la compagnie-division 3-1 et la liste des unités qui sont liées aux immeubles. Obtenir seulement les identifiants et les codes respectifs de chacun et inclure le nombre de pièces pour chacune des unités. Le tout trié en ordre d'immeuble.



Ces exemples ne montrent qu'un aperçu des possibilités avec OData. Veuillez consulter la documentation [OData - the Best Way to REST] pour plus de précisions.

WebHook

Voici pourquoi on peut vouloir utiliser un Webhook :

1. Réactivité en temps réel

Les webhooks permettent à une application d'être notifiée immédiatement d'un événement, sans avoir à interroger continuellement une API pour vérifier si quelque chose a changé. Cela est particulièrement utile pour des événements comme des notifications de paiement, des mises à jour d'état de commande, ou des modifications de données.

2. Efficacité

Contrairement au **polling** où un système interroge un serveur à intervalles réguliers (ce qui peut être inefficace et coûteux en termes de ressources), les webhooks n'envoient des données que lorsqu'un événement spécifique se produit. Cela réduit la charge du réseau et les coûts d'infrastructure.

3. Simplicité d'intégration

Les webhooks permettent une intégration facile entre des systèmes hétérogènes, car ils utilisent des requêtes HTTP standards (POST en général). De nombreuses applications SaaS (Software as a Service) fournissent des Webhooks pour notifier des événements importants à leurs utilisateurs ou à d'autres services.

4. Évolutivité

Les webhooks permettent à un système d'envoyer des notifications à plusieurs systèmes tiers sans avoir à traiter individuellement chaque demande ou à maintenir des connexions ouvertes en permanence, ce qui améliore l'évolutivité.

5. Réduction du délai de latence

Étant donné que les webhooks envoient des notifications dès que l'événement survient, le délai entre l'événement et sa prise en compte est très court. Cela améliore les performances globales et l'expérience utilisateur.

Voici quelques exemples ou les webhooks peuvent être utiles avec l'API Primmo:

- **Gestion des locations** : Dès qu'une unité se libère à une date précise, une notification peut être émise à une plateforme d'annonce en ligne des logements à louer.
- **Gestion documentaire** : Dès qu'un document est produit pour un locataire tel le bail, l'avis de renouvellement, un avis de retard, une notification peut être émise à un portail locataire afin d'avertir le locataire qu'un nouveau document est disponible pour lui.
- **Gestion des demandes de service** : Dès qu'une demande de service est créée, une notification peut être émise à une plateforme de répartition et d'assignation des tâches à exécuter.

En résumé, un webhook est utilisé pour assurer une communication rapide, efficace et automatisée entre deux systèmes, en particulier dans des contextes où la réactivité est cruciale.

Principe de fonctionnement

Le principe de fonctionnement normal d'un webhook repose sur un mécanisme simple d'événements déclenchés, permettant à une application (l'expéditeur) d'envoyer une requête HTTP à une autre application (le récepteur) en réponse à un événement particulier. Voici comment cela fonctionne en détail :

1. **Déclencheur d'événement**

- **Événement** : Un webhook est déclenché lorsqu'un certain événement survient dans l'application expéditrice. Par exemple, lorsqu'un utilisateur soumet un formulaire, fait un paiement, modifie un fichier, ou lorsqu'une nouvelle commande est créée sur une plateforme de commerce électronique.
- **Personnalisation** : L'expéditeur peut permettre à l'utilisateur de configurer des événements spécifiques qui déclencheront un webhook.

2. **Réception de l'URL cible**

- L'application qui veut recevoir les notifications (le récepteur) doit d'abord fournir une URL qui sera appelée par le webhook. Cette URL est souvent appelée "endpoint" ou "callback URL".
- **Configuration initiale** : Le récepteur configure l'application expéditrice pour l'informer de l'événement en enregistrant son URL auprès de l'application expéditrice.

3. **Envoi de la requête HTTP**

- **Requête HTTP** : Lorsque l'événement se produit, l'application expéditrice envoie automatiquement une requête HTTP (souvent une requête POST) à l'URL spécifiée par le récepteur.
- **Contenu de la requête** : Cette requête contient des données (payload) sur l'événement dans le corps de la requête. Ces données sont généralement envoyées au format JSON ou XML, mais cela peut varier en fonction de l'API.

4. **Traitement de la requête par le récepteur**

- **Réception des données** : Le serveur récepteur reçoit la requête HTTP et extrait les données envoyées par l'application expéditrice.

- **Action** : Le récepteur peut alors traiter ces données en fonction des besoins. Par exemple, enregistrer une transaction, envoyer une notification à un utilisateur, effectuer une autre requête à l'émetteur ou déclencher une autre action automatisée dans son propre système.

5. **Réponse du serveur récepteur**

- Le serveur récepteur doit envoyer une réponse HTTP (généralement un code 200 pour indiquer que la requête a été bien reçue et traitée). Si la requête échoue ou si le serveur ne répond pas correctement, l'application expéditrice peut réessayer d'envoyer la notification.

6. **Ré-essai automatique en cas d'échec**

- Si pour une raison quelconque l'URL de destination ne répond pas (erreur réseau, temps de réponse dépassé, etc.), l'application expéditrice peut tenter de réessayer l'envoi après un certain délai. Certains systèmes de webhooks permettent plusieurs tentatives sur une durée définie.

Récapitulatif schématique du fonctionnement :

1. Un **événement** se produit dans l'application expéditrice.
2. Une requête HTTP est envoyée à l'URL du webhook (URL du récepteur).
3. L'application réceptrice traite les données reçues.
4. L'application réceptrice renvoie un statut HTTP (ex : 200 OK).
5. En cas d'échec, l'application expéditrice peut tenter de renvoyer la requête.

Exemple d'utilisation :

Lorsqu'un locataire annonce son départ, l'unité devient alors disponible et le système déclenche un webhook afin d'informer un CRM de la disponibilité de l'unité et de mettre à jours ses information pour affichage.

Le fonctionnement d'un webhook repose donc sur la notification d'événements en temps réel via des requêtes HTTP envoyées à une URL configurée par l'application réceptrice. Il s'agit d'un moyen simple et efficace pour relier plusieurs systèmes entre eux sans requêtes répétitives (polling).

Configuration et utilisation des webhooks avec l’API Primmo

Afin recevoir une notification suite à un ajout, une modification ou à une suppression d’un élément dans une entité de Primmo/HAPI, un utilisateur s’abonner pour suivre ces changements de données. L’utilisateur doit d’abord se connecter à l’API pour obtenir un token puis ensuite effectuer les requêtes d’abonnements requises.

1 – Requête d’abonnement :

POST https://primmoapistaging.hopemservices.com/api/subscribe

Body:

```
{
   "ResponseUrl":"https://testing-website.com",
   "ResponseHeaders":"testing-website-token:TEST123456",
   "Entities": [ "building","unit" ]
}
```

Plusieurs entités peuvent être abonnées dans la même requête. *ResponseHeaders* peut être une chaine vide ou avoir la valeur *null*. Si plusieurs headers personnalisés sont nécessaires, les séparer par des « ; ».

2- Requête pour connaitre les entités suivies :

GET https://primmoapistaging.hopemservices.com/api/subscribe

Réponse:

[“Building”, “Unit”]

3- Requête de désabonnement :

POST https://primmoapistaging.hopemservices.com/api/unsubscribe

Body :

["building","unit"]

Liste d’une ou de plusieurs entités.

4- Webhook envoyé par le système :

{

"Id":1472,

"Entity":"TenantParking","

Status":2,

"DatabaseName":"DEV/ALEX",

"AdditionalInformationJson":"{\"tenantId\": 7535, \"parkingId\": 1445}"

}

Id : fait référence à l'ID de l'entité créée, modifiée ou supprimée.

Status : est le changement qui a eu lieu (1: UPDATE, 2: DELETE, 3: INSERT)

DatabaseName : est le nom du client ou de la base de données.

AdditionalInformationJson : n'a pas toujours une valeur. Seulement les tables relationnelles qui sont suivi par le système de webhooks contiennent une valeur.

5- Entités où l'abonnement est disponible :

La page Swagger de l'api met à jour l'information des entités où l'abonnement est disponible. En date d'octobre 2024, une vingtaine d'entités sont disponibles.

Pour en nommer quelques-uns : Tenant, Unit, TenantService, ServiceCall, ServiceCallDetail, Invoice, Supplier, TenantParking

Q&R

Q1. Je dois fréquemment obtenir la liste des locataires et ce aux 15 minutes afin de mettre à jour un système tiers. Il y a beaucoup de données qui transitent lors de l'appel à l'API. Y a-t-il moyen de réduire le nombre le volume de données ?

R1. Il est possible d'optimiser ce processus en le rendant plus efficace avec différentes options, lesquelles peuvent être utilisées en combinaison.

a. Avec OData, il est possible d'utiliser la clause $select afin de ne sélectionner que les informations utiles.
b. Avec OData, il est également possible d'utiliser la clause $where afin de réduire et de filtrer efficacement les données.
c. En utilisant les capacités *webhooks* de l'API, vous êtes notifiés des changements se produisant dans Primmo. Vous pouvez donc mettre à jour votre système tiers seulement lorsque des changements se produisent.

Q2. Quelle est l'entité me permettant de récupérer la liste générale des services afin de mettre à jour mon CRM?

R2. Le document intitulé « Documentation Hapi Primmo V##.pdf » est utile à l'identification des entités et de leurs propriétés.

Q3. Comment faire un premier appel à l'API ?

R3. Il est recommandé d'utiliser Postman ou un outil similaire afin de tester la connectivité à notre API.

Notre collection Postman n'est pas encore disponible pour le moment, mais voici la structure des appels :

3- Obtenir le *token*

L'URL est **{url}/api/Token**

La clé API doit être passée comme valeur dans les *Headers* avec la clé **x-api-key**

Le corps du message (*body*), au format json, doit contenir utilisateur et mot de passe :

```
{
  "Password":"myPassword",
  "UserName":"myUserName"
}
```

Le *token* sera retourné dans le corps du message, et réutilisé à la prochaine étape.

4- Obtenir la liste des immeubles (par exemple)

L'URL est **{url}/api/Buildings**

Le token (Bearer) doit être passé dans les Headers avec la clé **Authorization.** La valeur doit être préfixée de "Bearer "

ex : "Bearer eyJhbGciOiJIUzI1NiIsInR5c……"

Puisque le *token* expire après 24 heures, il est nécessaire de générer un nouveau *token* après son expiration. Lors de l'utilisation de Postman, il est recommandé de configurer des variables pour gérer automatiquement les valeurs de **x-api-key** et du token. Cela simplifie le processus de gestion des authentifications, car les variables permettent de réutiliser ces informations dans plusieurs requêtes sans avoir à les mettre à jour manuellement à chaque fois. Cette approche garantit une meilleure flexibilité et efficacité dans les tests et appels API, tout en réduisant les erreurs liées à la manipulation des clés et des *tokens* d'authentification.

Cela devrait être suffisant pour être en mesure de faire les premiers appels à l'API.

Il n'y a pas de refresh token. Vous avez simplement besoin de redemander un nouveau token d'accès après 24h. Avec le token reçu vous pouvez appeler tous les autres points d'accès en ajoutant dans votre entête Bearer et ce token. Il y a qu'un seul token et celui-ci vous donnes accès aux entités.

Q4. Comment obtenir la liste des compagnies-divisions?

R4. Le point d'accès «api/LedgerDivisions» répond à ce besoin.

Q5. Est-ce qu'il est possible d'ajouter des informations dans Primmo via l'API?

R5. Certainement. Cependant, il n'est possible actuellement que pour quelques entités telles : les locataires, les services liés au bail, la location de stationnements et rangements, les appels de service, et quelques autres encore. La documentation Swagger peut vous renseigner à ce sujet

Q6. Est-ce que l'information est validée lorsque les données sont ajoutées par l'API?

R6. Les données ajoutées sont soumises aux mêmes règles de validation que si elles étaient ajoutées par Primmo.

Q7. Le code d'occupation de l'unité 101 indique qu'il est loué alors qu'il deviendra disponible dans moins de 30 jours.   Comment gérer cette situation?

R7. Le code d'occupation informe du statut actuel de l'unité et non de son statut passé ou futur. Il est préférable d'utiliser le point d'accès « api/Units/ForRent » pour obtenir la liste des unités disponibles à l'intérieur d'une plage de dates donnée. De plus, ce point d'accès donne accès à plusieurs propriétés et caractéristiques des unités.

Q8. Je vois dans la documentation que le point d'accès «api/FinancialStatements » est disponible dans l'API mais j'obtiens l'erreur **403 forbidden** lors de l'appel. Pourquoi ?

R8. Il est probable que le client n'ai pas fait l'acquisition du module requis dans lequel de point d'accès est inclus. Veuillez en informer le client pour qu'il contacte notre département des ventes afin d'identifier les besoins du client.

Q9. Un bail de 36 mois est signé avec un locataire avec des augmentations préprogrammées de 20$ mensuellement par année. Ex : 1000 le premier mois, 1020$ l'année suivante puis 1040$ l'année suivante. Comment traiter cette information?

R9. Utiliser la commande POST du point d'accès api/LeasePriceChanges pour chaque augmentation. Le traitement de facturation s'occupera d'activer le montant loyer voulu à la date précisée.

Q10. Je veux appliquer un rabais de 50$ sur les 2 premiers mois de loyer. Comment faire ?

R10. Utiliser la commande POST du point d'accès api/TenantServices afin d'ajouter un rabais couvrant la plage désirée.

Q11. Certaines informations ne sont pas disponibles encore dans l'API. Est-ce qu'une *roadmap* est disponible? Est-il possible de prioriser certains développements?

R11. Notre roadmap n'est rendue public que pour la prochaine version puisque l'évolution de notre API est basée sur de multiples axes de développements, lesquels évoluent rapidement au fil du temps. Vous pouvez contribuer au développement de la solution soit en faisant des demandes d'améliorations (sans aucun engagement de notre part), soit en demandant des travaux explicites à notre équipe de développement, lesquels sont des travaux facturables.

Q12. Nous récupérons une unité qui apparait aussi dans la liste des unités non-disponibles via api/unavailableUnits. Cependant, les dates d'indisponibilités étant passées, cette unité devrait être disponible à la location, mais ce n'est pas le cas. Comment obtenir les unités disponibles à la location?

R12. Afin d'obtenir la liste des unités disponibles, le point d'accès api/Units/ForRent vous permettra d'obtenir précisément cette information. Ce point d'accès tient compte à la fois des locations en cours de même que de la non-disponibilité des unités.

Q13. Nous voulons ajouter une location d'un stationnement à un locataire mais le système refuse. Pourtant la date de début indiquée est ultérieure à la fin de la location précédente à un autre locataire. Quelle est la raison?

R13. Lorsque la location précédente du stationnement est renouvelable, le stationnement est conservé pour le locataire tant qu'une date de départ n'est pas inscrite à la fiche du locataire.

Q14. Serait-il possible d'avoir un compte de service API dédié pour cette intégration (plutôt qu'un compte utilisateur existant) ?

R14. Tous les Clés et comptes utilisateur sont dédiés. Normalement, une clé par intégrateur et un user_passord par intégrateur.

Q15. Y a-t-il des restrictions réseau à prévoir (VPN, IP allowlist, etc.) ? Y a-t-il une limite de requêtes API ou une taille maximale par requête ?

R15. Non. Il est fortement recommandé d'utiliser les options de requête ODATA et les webhooks pour réduire la taille de vos requêtes et l'utilisation des ressources. Nous préconisons fortement l'utilisation des webhooks dès le début du projet pour éviter un usage inutile de nos ressources. Si les webhooks ne sont pas utilisé nous pourrions être dans l'obligation d'établir des limites aux requêtes.

Q16. Existe-t-il un champ de type ModifiedDate / UpdatedAt pour permettre une synchronisation incrémentale ?

R16. En général, non, mais plusieurs entités contiennent des champs de dates en références aux transactions ou statuts. Il y a quelques entités qui contiennent des dates de modification. Consulter la documentation Swagger pour plus de précisions.

Q.17 Comment fonctionne votre pagination (OData : $top/$skip ou @odata.nextLink) ?

R17. Voir la documentation ci-haut pour les options de requête OData disponible actuellement. D'autre options de requête seront disponibles dans le futur en fonction des demandes des clients. Le produit est en constante évolution.

Q18. Quel est l'identifiant unique et stable pour chaque entité (immeubles, unités, locataires) ?

R18. Voir la doc ci-jointe. Il y a généralement des clés primaire ''ID'' dans les entités. Voir aussi les Dto Swagger UI.

Q19. L'API est-elle bien bidirectionnelle (lecture et écriture) ?

R19. Partiellement, certaines entités ne sont pas encore couvertes pour la création/modification.

Q20. Peut-on faire des push API pour la création / mise à jour de factures ?

R20. Création seulement pour le moment