Échanges avec un logiciel de gestion de demandes «métier»

Un cas habituel de logiciel tiers est celui qui traite des demandes «métier», par exemple des signalements de problèmes sur la voiries ou des demandes d'intervention concernant la gestion des eaux.

Un tel logiciel être connecté avec Publik afin que :

- un usager puisse envoyer une demande (un signalement) vers le logiciel
- par la suite, qu'il puisse être informé de l'avancement de sa demande (prise en charge, traitement en cours, etc.)

Pour mettre en place cette connexion, le logiciel doit exposer un certains nombres de webservices à disposition de Publik. Cette documentation les décrit.

Principes des échanges

Publik créé la demande dans le logiciel tiers : les données de la demande et les documents liés si nécessaire.

Publik interroge ensuite le logiciel pour connaître le statut de la demande. Ce statut est éventuellement accompagné d'informations complémentaires.

Selon les informations reçues, Publik informe l'usager de l'avancement de sa demande, par courriel, SMS, message à l'écran ou tout autre moyen.

Publik répète cette interrogation toutes les 6 ou 12h, jusqu'à ce que la demande atteigne un statut «final». Les échanges la concernant s'arrêtent alors.

Création d'une demande dans le logiciel tiers

Une fois la demande saisie par l'usager, Publik l'envoie dans le logiciel métier. Pour cela, un appel HTTP POST est fait vers une URL proposée par le logiciel, dans notre exemple https://logiciel.tiers/api/creation-nouvelle-demande, mais tout autre format d'URL est possible.

Cet appel POST envoie un dictionnaire JSON, à plat, contenant une suite de clés-valeurs :

```
POST https://logiciel.tiers/api/creation-nouvelle-demande
Content-Type: application/json
Accept: application/json

{
  "info1": "valeur1",
  "info2": "valeur2",
  "info3": "valeur3",
  "info4": "valeur4",
  ...
}
```

Noter que le formulaire n'est pas envoyé *tel quel*, tel que saisi par l'usager : certains champs peuvent être modifiés, adaptés, transformés ou normalisés. Par exemple, si l'usager a choisi la ville "Grenoble" dans la liste, c'est son code INSEE qui peut être envoyé dans le logiciel (cf plus bas la gestion des référentiels). C'est donc au logiciel tiers de décrire l'ensemble des données dont il a besoin pour créer une demande : le formulaire Publik sera adapté pour les obtenir de l'usager, et la traduction technique précise sera faire lors de l'appel HTTP POST.

Attention : par défaut pour chaque "valeurX", Publik ne sait envoyer que des chaînes de caractères. Ainsi, si des nombres doivent être transmis, ils le seront sous forme de chaîne de caractère. D'autres types de données peuvent être possibles, mais il convient alors de vérifier que Publik peut les gérér.

En retour, Publik attend un dictionnaire avec deux entrées :

- err : doit être à 0 (le nombre entier 0) en cas d'absence d'erreur
- data : contient toute les données nécessaires à la suite du traitement. Ça sera en général la référence de la demande créée dans le logiciel, référence qui sera utilisée par Publik pour les communications ultérieures relative à cette demande.

Exemple de réponse attendue :

```
POST https://logiciel.tiers/api/creation-nouvelle-demande
(... dictionnaire JSON de la demande à créer ...)

200 OK
Content-Type: application/json

{
  "err": 0,
  "data": {
     "numero": "42",
     "url": "https://logiciel.tiers/api/demande/42/",
     "statut": "demande créée",
     "datetime": "2021-09-09T15:20:12"
  }
}
```

Publik va stocker toutes ces informations dans la demande. L'élément important et nécessaire ici est numero, qui sera la référence dans le logiciel.

En cas de problème, la réponse sera du même format mais avec un err à 1 (ou toute autre valeur différente du nombre 0) : lire la section sur la gestion de erreur plus bas dans ce document.

Ajout de document sur une demande créée

Une fois la demande créée, Publik peut y joindre les documents. Les documents sont envoyés en dehors de la création de la demande, et un par un : l'objectif est d'éviter tout engorgement, car il s'agit souvent de documents volumineux qui peuvent subir un filtrage par un équipement réseau entre Publik et le logiciel.

Pour chaque document, une requête HTTP POST est effectuée qui contient une référence à la demande créée. Par exemple :

```
POST https://logiciel.tiers/api/document-pour-demande/42/
Content-Type: application/json
Accept: application/json

{
  "document": {
     "filename": "nomdufichier.pdf",
     "content_type": "application/pdf",
     "content": "Y2VjaSBlc3QgdW4gZXhlbXBsZSBkZSBiYXNlNjQK..."
  },
  "type": "passeport",
}
```

Le fichier est sérialisé dans une clé document. Le contenu du fichier est envoyé dans content encodé en base64, le nom du fichier est filename et son type MIME est dans content_type. Ce format filename+content_type+content est imposé par Publik et n'est pas modifiable.

Dans l'exemple ci-dessus, un type est indiqué qui permet au logiciel métier de savoir de quel document il s'agit. D'autres informations peuvent accompagner le document, selon ce qui est nécessaire au logiciel.

Le numéro de la demande pour laquelle le fichier est envoyé (ici 42) peut être indiqué :

- soit dans l'URL du POST, comme dans l'exemple
- soit dans la query-string, on aurait ici /api/document-pour-demande/?demande=42
- soit dans le dictionnaire JSON

En retour de cet appel HTTP POST, Publik attend un dictionnaire avec au moins une entrée err à 0 (le nombre entier), et éventuellement des informations supplémentaires dans une clé data, par exemple :

```
200 OK
Content-Type: application/json

{
```

```
  "err": 0,
  "data": null
}
```

Comme pour la création de la demande, en cas d'erreur la réponse sera identique mais avec err à 1 (ou toute autre valeur différente du nombre 0) : voire la section sur la gestion d'erreurs plus bas dans ce document.

Remontée du statut d'une demande dans Publik

Une fois la demande créée dans le logiciel, elle va y être traitée.

Publik va régulièrement interroger son statut, par exemple toutes les 6 heures, en effectuant une requête HTTP GET telle que :

```
GET https://logiciel.tiers/api/statut-demande/42/
Accept: application/json
```

Le numéro de la demande peut être dans l'URL comme dans cet exemple, ou dans la *querystring* et on aurait alors par exemple /api/statut-demande/?demande=42.

En retour de cet appel HTTP GET, Publik attend un dictionnaire avec deux entrées err et data : c'est dans data que sont les informations de statut de la demande.

Exemple de réponse attendue :

```
Content-Type: application/json

{
  "err": 0,
  "data": {
     "statut": "traitement-en-cours",                          # code du statut
     "statut_label": "Demande en cours de traitement"          # nom affiché à l'usager
     "commentaire": "Votre demande sera traitée le 15/01/2021"
  }
}
```

Pour savoir réagir dans toutes les situations, Publik devra connaître **tous** les statuts possibles, pour savoir ce qu'il doit faire de son côté : informer l'usager, clôturer la demande, etc.

Exemples de statuts que l'on peut imaginer (mais c'est le logiciel qui spécifiera la liste exacte selon son «métier») :

- refus
- traitement-en-cours
- cloture

Publik interroge régulièrement le statut de la demande, jusqu'à obtenir un statut final (par exemple "clos" ou "refus"). Le délai entre chaque interrogation est paramétrable dans Publik, c'est souvent 6h ou 12h, et ce pour chaque demande qui n'est pas encore clôturée.

Commentaires lors du traitement dans le logiciel tiers

Pendant le traitement de la demande, les agents peuvent éventuellement indiquer des commentaires destinés à l'usager.

Pour faire cela, lors de chaque interrogation du statut de la demande, Publik vérifie si le commentaire a changé. Si oui, alors il le communique à l'usager (par mail, sms, message à l'écran, selon l'action configurée dans Publik).

Système en mode «push» : déclenchements par le logiciel tiers

Dans ce qui est décrit ci-dessus, c'est Publik qui vient régulièrement demander au logiciel tiers le statut des demandes en cours. Cela présente plusieurs inconvénients :

- si 300 demandes sont en cours, Publik va faire 300 requêtes
- le délai entre chaque requête sur une demande sera de plusieurs heures (souvent 6 et 12) et ne permet pas donc pas réaction rapide du Publik

- si la demande a changé plusieurs fois de statut dans l'intervalle, Publik peut avoir raté les étapes transitoires cela peut compliquer la compréhension par l'usager de l'avancement de sa demande

Si le nombre de de demandes est important ou si la réactivité doit être grande, c'est plutôt le logiciel tiers qui doit informer de l'avancement des demandes.

Pour cela, le logiciel tiers peut déclencher des appels «_trigger_» sur Publik, sous forme de requête POST, selon ce format :

```
POST https://demarches.example.net/<slug-demarche>/<numero-de-demande>/jump/trigger/<declencheur>/
{
  "information": ...
}
```

Cet appel va demander à Publik que la demande de type <slug-demarche>, numéro <numero-de-demande>, effectue un saut nommé <declencheur>. Ce saut est programmé dans Publik, dans le workflow de la démarche concernée. Au passage, le logiciel pourra fournir des informations supplémentaires dans un dictionnaire JSON, qui seront enregistrée dans la demande Publik. Ces données pourront ensuite être exploitées dans le workflow, par exemple pour alimenter une donnée de traitement.

Plus de détails sur ce principe de traitement : https://doc-publik.entrouvert.com/dev/wcs/api-webservices/traitement-d-un-formulaire/

Ce mécanisme impose que le logiciel tiers détermine les URL à appeler. Pour cela, il devra connaître les codes <declencheur> possibles, et les appeler au moment opportun. Il doit aussi avoir reçu <slug-demarche> et <numero-de-demande> dans le JSON de la création de la demande.

Enfin, en mode «push», le logiciel tiers doit tracer (loguer) les appels qu'il effectue. Il doit savoir retenter un appel qui a échoué sur une erreur non fatale (erreur réseau, DNS, code HTTP 5xx, etc.). Il doit savoir lever une alerte en cas d'erreur fatale afin de pouvoir déboguer les problèmes le plus rapidement possible. (Pour information, comme pour toute requête, Publik ne logue que les métadonnées, donc ni le contenu des appels reçus, ni celui de la réponse envoyée.)

Référentiels, listes de choix

Le formulaire présenté à l'usager par Publik comporte souvent des champs de type listes. L'usager doit choisir un ou des éléments dans ces listes. Ces choix sont nécessaires à la création de la demande dans le logiciel métier, par exemple le type d'intervention, la nature du signalement, etc.

Le logiciel tiers doit fournir à Publik le contenu de ses listes. Comme exemple on imagine le cas d'un formulaire de demande d'intervention dans une école située sur le territoire d'une agglomération : le formulaire affichera les listes suivantes :

- les villes
- les établissements concernant la ville sélectionnée

Pour chaque liste, Publik interroge un webservice de référentiel, dans cet exemple on aurait :

- GET https://logiciel.tiers/api/referentiel/villes
- GET https://logiciel.tiers/api/referentiel/etablissements

La réponse à un référentiel est un document JSON de type dictionnaire, avec deux entrées :

- err : un code d'erreur, égal à 0 (le nombre entier 0) quand tout se passe bien (voir en fin de document pour la gestion des erreurs)
- data: la liste des items du référentiel, chaque item étant un dictionnaire contenant :
  - id : identifiant unique de l'item
  - text: texte affiché à l'usager, typiquement dans une liste affichée sur le formulaire
  - toute autre information utile à afficher à l'usager ou à envoyer ensuite au logiciel dans la demande

Exemple d'une réponse simple pour la liste des villes, avec juste id et text pour chacune :

```
GET https://logiciel.tiers/api/referentiel/villes

200 OK
Content-Type: application/json

{
  "err": 0,
  "data": [
```

```
    {
      "id": "38185",
      "text": "Grenoble",
    },
    {
      "id": "38544",
      "text": "Vienne"
    }
    (... les autres villes ...)
  ]
}
```

Mais on peut aussi ajouter d'autres détails sur chaque entrée, par exemple le code postal principal de chaque ville :

```
GET https://logiciel.tiers/api/referentiel/villes

200 OK
Content-Type: application/json

{
  "err": 0,
  "data": [
    {
      "id": "38185",
      "text": "Grenoble",
      "code_postal": "38000",
    },
    {
      "id": "38544",
      "text": "Vienne"
      "code_postal": "38200",
    }
    (... autres villes ...)
  ]
}
```

À noter que les éléments des listes doivent être triés dans l'ordre à afficher aux usagers, Publik ne re-trie pas les listes qu'il reçoit.

Usage des données reçues

Publik affiche sur le formulaire la liste des text, dans l'exemple ci-dessus ça sera Grenoble, Vienne, etc. Quand l'usager fait son choix dans cette liste, alors Publik dispose de toutes les informations liées à ce choix. Par exemple si l'usager choisi Vienne, Publik connaitra les valeurs suivantes :

- le nom de la ville choisie : "Vienne" (clé text)
- le code INSEE de cette ville : "38544" (clé id)
- et son code postal : "38200" (clé code_postal)

Toutes ces informations sont stockées et donc partie de la demande Publik. Elles peuvent toutes être utilisées pour communiquer ensuite avec le logiciel tiers lors de la création de la demande.

Ainsi, au lieu d'envoyer le nom "Vienne" au logiciel tiers qui devrait retrouver à quelle ville sa correspond dans sa base, Publik peut lui envoyer directement le code INSEE de la ville concernée "38544". Ce partage de référentiels permet donc une communication plus efficace entre Publik et le logiciel.

Filtrage des éléments d'un référentiel

Pour certaines listes, on voudra les restreindre en fonction de certains critères. Dans notre exemple on cherche à afficher les établissements d'une ville en particulier. Pour faire cela, Publik ajoute un critère dans l'URL lorsqu'il interroge le référentiel des établissements : GET https://logiciel.tiers/api/referentiel/etablissements?code_ville=38000

La réponse obéit strictement au même format que pour la liste complète des établissements, mais seuls les établissements de la ville concernée sont présents dans data.

Liste en mode auto-complétion

Quand une liste contient plusieurs dizaines d'éléments, elle est trop longue à afficher dans un formulaire. Publik peut en proposer l'affichage en mode *auto-complétion* : l'usager doit taper quelques lettres et Publik affiche les éléments de la liste qui les contiennent.

Pour cela, le webservice référentiel concerné doit réagir à deux paramètres supplémentaires : q pour filtrer en fonction de ce que l'usager a tapé, et id pour remonter les informations du choix fait par l'usager.

- GET https://logiciel.tiers/api/referentiel/villes?q=gren : doit renvoyer la liste des villes qui contiennent gren, les 4 lettres tapées par l'usager pour sa recherche. Les éléments de la liste doivent être renvoyés selon l'ordre le pertinence (les plus pertinents en premier).

- GET https://logiciel.tiers/api/referentiel/villes?id=38000 : doit renvoyer une liste ne contenant qu'un seul élément, à savoir la ville ayant le code 38000

Comme pour une requête filtrée, la réponse obéit strictement au même format que pour la liste complète : un dictionnaire avec err à 0 (le nombre entier 0) et data qui contient la liste des éléments.

Format des réponses en cas d'erreur

Lorsque Publik envoie des données mal formatées

Si les données envoyées par Publik ne sont pas dans un format attendu par le logiciel, celui-ci doit renvoyer une erreur avec le détail des incohérences constatées. Le retour est toujours un dictionnaire JSON avec deux entrée "err" et "data", mais cette fois err doit être différent de 0. Attention, erreur fréquente : si err contient la chaine de caractère "0" alors ça sera considéré comme un code d'erreur. Il faut utiliser le nombre entier 0.

Le message technique peut être en anglais (par exemple un retour d'erreur SQL) mais il doit permettre à un technicien de comprendre quel champ est en cause, et si possible le type d'erreur constaté (valeur impossible, mauvais type de donnée, …). Le message doit être contenu dans une clé "err_desc" (description de l'erreur). Si c'est utile, un code type d'erreur peut être retourné dans "err_class".

Le code de retour HTTP peut être 400, mais ce n'est pas obligatoire, le plus important est "err" différent de 0.

Par exemple si Publik envoie une demande avec une mauvaise information "foo" :

```
POST /api/creation-demande
{
  "foo": "trois"
}

400 Bad Request
Content-Type: application/json

{
  "err": 1,
  "data": null,
  "err_desc": "valeur de foo non acceptée, doit être un entier",
  "err_class": "bad-request"
}
```

Autres erreurs

Pour toutes les autres erreurs, par exemple si une écriture SQL ne se fait pas, ou qu'un autre problème technique survient lors de l'exécution du webservice par le logiciel, ce dernier doit tout de même répondre une réponse JSON (afin que Publik sache qu'il a bien interrogé le webservice mais que celui-ci a eu un problème interne).

Typiquement, il ne fait jamais renvoyer une erreur de type 500 ou une "traceback".

Donc, même en cas d'erreur :

- Le code HTTP de retour doit toujours être 200 (c'est-à-dire « l'échange avec le webservice a eu lieu et a voici sa réponse ») modulo l'exception du code 400 indiquée plus haut en cas de mauvais format d'interrogation du webservice.
- Le contenu du retour doit toujours être un dictionnaire JSON
- La clé "err" dans le JSON doit être différente de 0 ; très fréquemment on indique juste le nombre 1

- La clé "data" peut être vide ou inexistante ; on évitera de la valoriser pour ne pas engendrer de confusion

Pour aider à la compréhension de l'erreur, on peut ajouter :

- une clé "err_desc" qui décrira l'erreur, si possible en français. Elle est destinée à être affichée à un administrateur fonctionnel ou un technicien de Publik qui pourra comprendre le soucis
- une clé "err_class" contenant un identifiant de l'erreur, qui peut être par exemple utilisé pour que Publik gère automatiquement une nouvelle tentative si l'erreur est conjoncturelle (non fatale).

Par exemple en cas d'impossibilité de joindre le SGBD :

```
GET /api/status-demande?id=42

200 OK
Content-Type: application/json

{
  "err": 1,
  "data": null,
  "err_desc": "table form_evolutions inaccessible",
  "err_class": "sql-error"
}
```

Sécurisation de la communication entre Publik et les webservices du logiciel

Tous les échanges auront lieux en HTTPS avec un certificat valide. Au cas où le certificat est émis par une autorité de certification locale, il faut fournir le certificat de l'autorité (clé publique).

L'accès peut être reservé aux adresses IP d'hébergement de l'instance Publik qui effecture les requêtes.

Enfin, une authentification de type HTTP Basic peut être ajoutée en sécurité supplémentaire.