Contenu Masquer

1 Objet
2 Schéma général d'architecture:
3 Sous-systèmes
3.1 Sous-système de backoffice
3.1.1 Fonctionnement sur mobiles
3.2 Sous-système de FrontOffice
3.2.1 Formulaires d'interaction
3.2.1.1 Avantages d'utiliser des formulaires externes
3.2.1.2 Design des formulaires
3.2.1.3 Formulaires
3.2.1.3.1 Contexte
3.2.2 Conversations et Notifications
3.2.2.1 Types de mail
3.2.2.2 Extraction des pièces attachées
4 Annexes
4.1 Exemple de script accompagnant un formulaire Google
4.2 Limites d'Oracle Email Delivery Service pour Free Tier

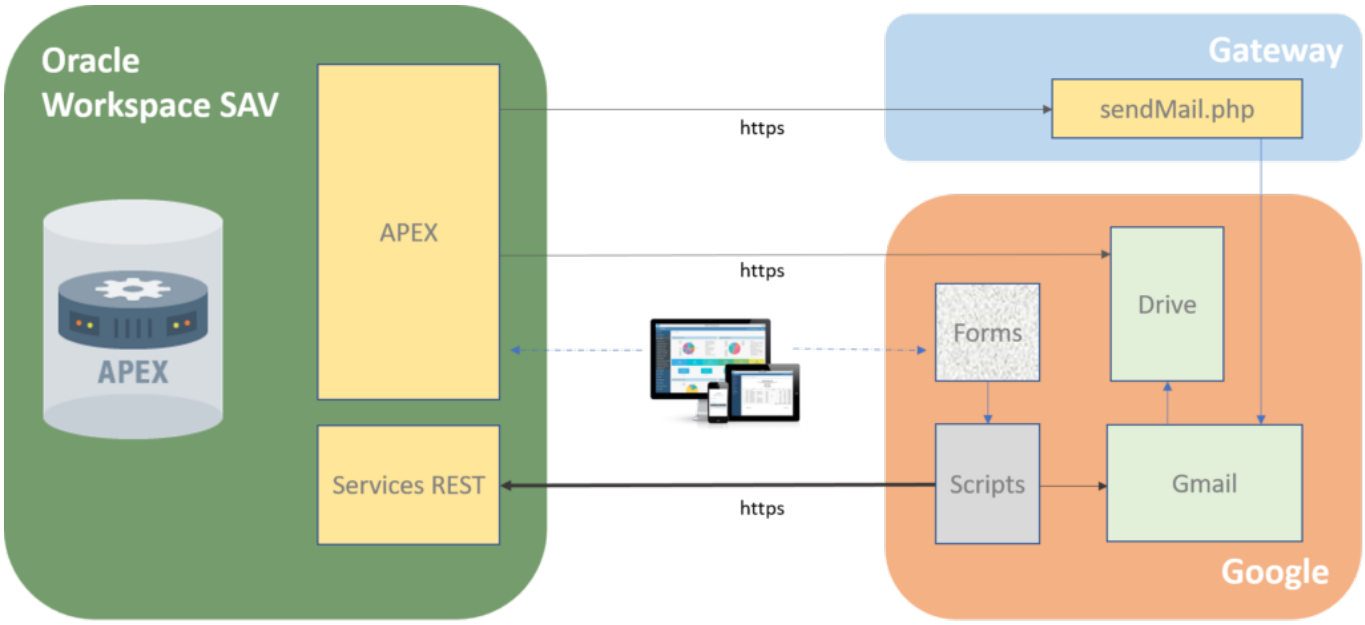
Objet

C'est l'histoire d'une application de SAV (service après vente) réalisée initialement avec Oracle APEX et complétée ultérieurement avec des module de Google Workspace. Cela a permis de constituer deux sous-systèmes étanches, l'un destiné au backoffice pour les techniciens chargés du support et l'autre destiné aux interactions avec les clients du SAV.

Oracle APEX est utilisé en version *free tier* (avec des limitations mais … gratuit !) dans la mesure où il y a un petit nombre d'analystes (inférieur à cinq dans le contexte de mon projet) et donc de faibles contraintes de concurrence d'accès et de charge CPU.

Toutes les interactions avec les clients sont gérées avec des formulaires Google Forms, des scripts, des envois/réceptions de mail et du stockage de pièces attachées sur Google Drive.

Schéma général d'architecture:

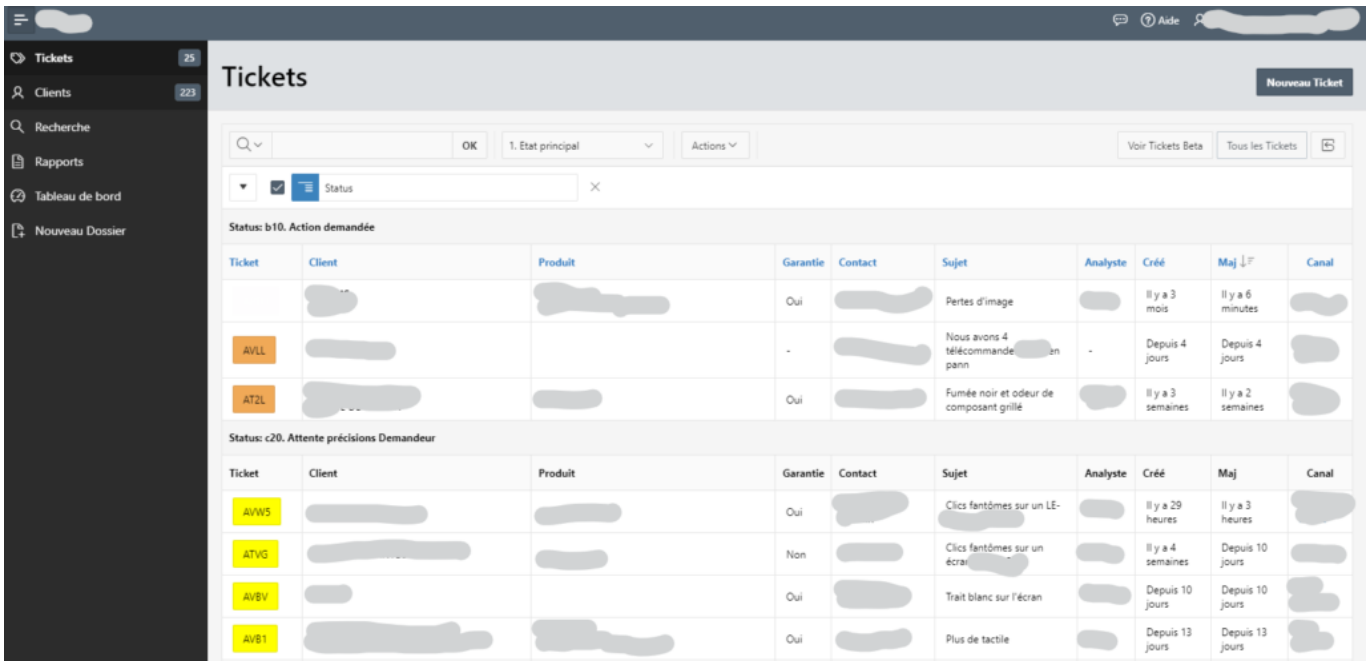


Sous-systèmes

Sous-système de backoffice

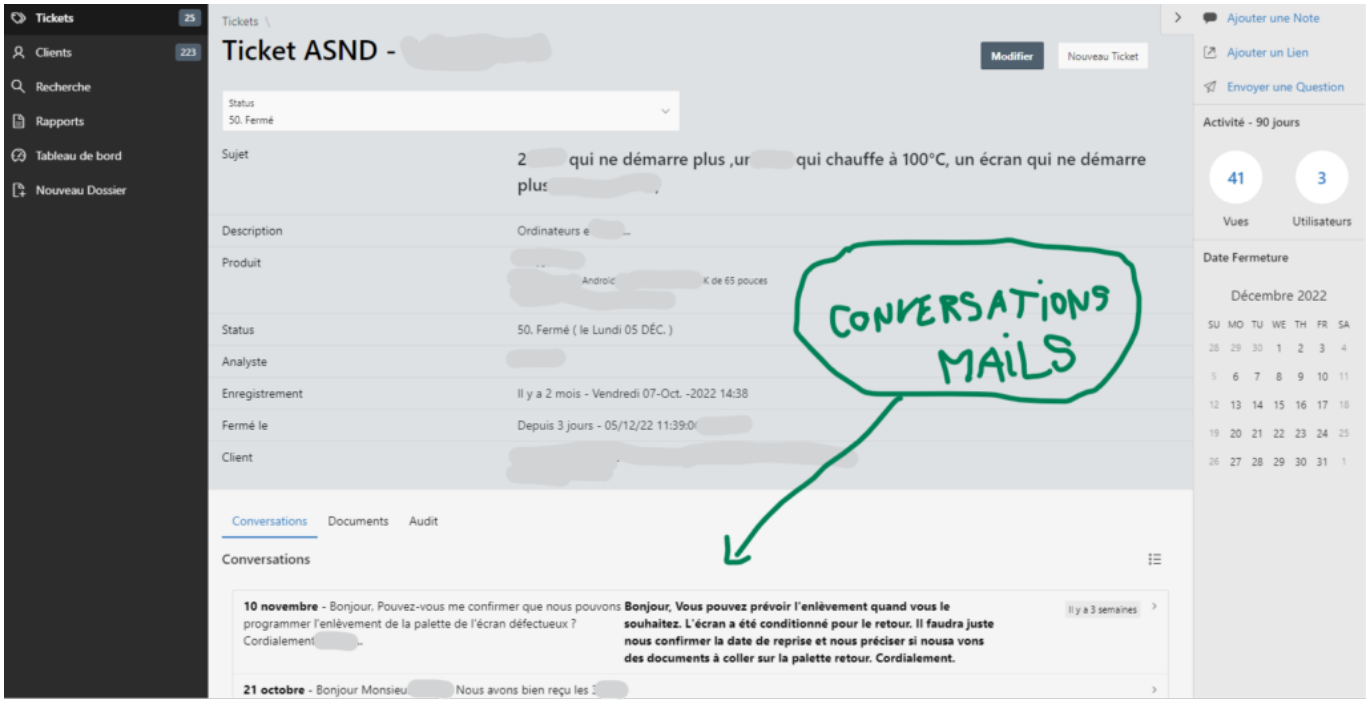
Je suis parti d'une application standard qui était fournie avec les précédentes versions d'Oracle APEX et qui s'appelait *Incident Tracking*. Puisque cette application n'est plus disponible dans les version récentes d'APEX, je l'ai récupérée depuis une version 18.x. Cela m'a permis de réutiliser le modèle de données qui était de qualité et de proposer

un démarrage rapide. Mais au fur et à mesure, il s'est avéré que l'écart fonctionnel était plus grand que prévu. Rétrospectivement, je serais plutôt parti d'une feuille blanche, tout en conservant le modèle de données.



Liste des tickets ouverts

L'allure générale est très proche de celle d'origine, mais il y a eu beaucoup de règles métiers spécifiques qui ont été rajoutées et qui ont fait bouger également le modèle de données. Toutes les informations du clients sont récupérés, en particulier, depuis un ERP par appel d'un service Web. Pour le reste, il s'agit d'une application classique de gestion.



Résumé d'un ticket

Les éléments de conversation entre le(s) technicien(s) et le client sont rassemblés dans un onglet *Conversations* et sont issus du sous-système d'interactions, tel que décrit plus loin.

L'onglet *Documents* contient la liste des liens vers *Google Drive* dans le cas où le client aurait envoyé des fichiers descriptif du problème sous forme de photos et vidéos.

J'ai profité des nouveautés des version récentes d'APEX, en particulier la recherche par facettes, pour améliorer l'ergonomie et j'ai rajouté la possibilité de se connecter avec son compte Google Workspace puisque chaque technicien en disposait.

Fonctionnement sur mobiles

Très peu de modifications ont été nécessaires pour obtenir un fonctionnement correct sur mobile. J'ai simplement utilisé des classes (hidden-xxs-down et hidden-xs-up) pour

switcher sur une présentation plutôt qu'une autre au sein d'une même page. C'est typiquement le cas pour les listes de type « Interactive Grid » qui auraient imposé du scrolling sur un petit écran et auxquelles j'ai substitué des *list View*s.
Les autres composants Oracle APEX s'adaptent automatiquement à la géométrie du *device*.
J'en ai profité pour mettre un peu de PWA pour gagner un peu d'espace sur l'écran du mobile et proposer une icone plus naturelle qu'un raccourci du Navigateur.

Sous-système de FrontOffice

Ce deuxième bloc est destiné aux interactions avec les clients finaux. Il repose sur l'emploi de plusieurs modules de la suite *Google Workspace*. La compagnie disposait déjà d'un abonnement à ce service pour la messagerie et il n'a pas été nécessaire de souscrire à des options complémentaires.

Formulaires d'interaction

Concevoir et mettre à disposition des formulaires d'ouverture de ticket ou de demande de statut directement sous Oracle APEX aurait entrainé la communication de l'url du *backoffice* auprès de chaque client. Un contournement éventuel aurait consisté à utiliser le *Load Balancer* disponible dans le *tenant* en reverse Proxy et d'y placer des règle de routage en fonction de l'application à utiliser. A l'époque, je n'étais pas familier avec la technique pour le faire et, de toute manière, cela aurait posé le problème de la charge du système car si je connaissais le nombre d'utilisateurs internes (les analystes), je ne connaissais pas le nombre de clients qui se connectent à un instant donné. Un nombre élevé de connexions aurait entrainé la fermeture arbitraire de sessions déjà ouvertes, puisqu'il y a un pool limité dans le cas de Oracle

Free Tiers, et cela aurait pénalisé l'accès par les technicien comme le laisse supposer l'extrait suivant des limitations:

The HTTP interface of Always Free APEX Service is rate limited to restrict the number of simultaneous service users. Approximately 3-6 simultaneous users can be supported across all of APEX, Oracle REST Data Services, and SQL Developer Web

J'ai donc fait le choix d'un sous-système complétement séparé, responsable uniquement des interactions avec les clients. Google Forms était un choix disons … spontané, puisqu'il était question de formulaires.

Tous les formulaires ont été conçus avec Google Forms. Autant le préciser de suite, ce service souffre de beaucoup de limitations car il est impossible, par exemple, de proposer une liste de valeurs dynamique. Il n'est pas possible non plus d'afficher un champs en mode *read-only*. Il a fallu donc s'accommoder de nombreuses limitations. C'était le prix à payer pour avoir l'esprit tranquille quant aux considérations de sécurité et de montée en charge. Il existe bien sûr des plugins spécialisés pour Google Forms mais ils sont payants ou bien apportent une publicité intempestive.

Avantages d'utiliser des formulaires externes

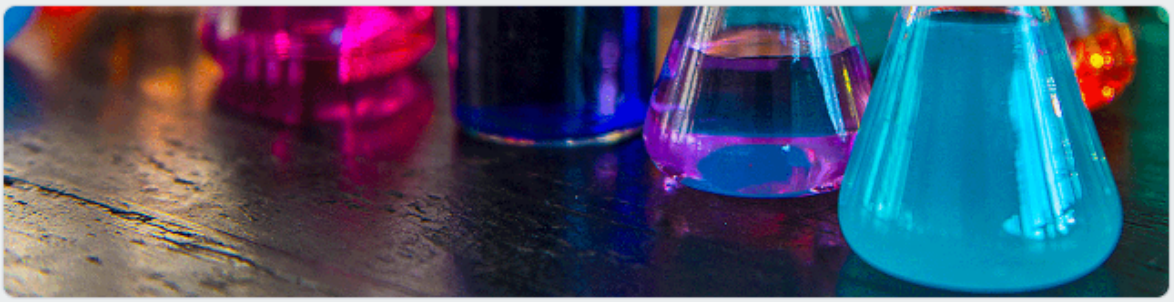
Le client n' a jamais connaissance de l'existence de l'url de *backoffice*. Cela évite une publicité non désirée du système. Tout ce qu'il voit, c'est l'interface Google Forms qui, je le suppose, est suffisamment protégée contre des attaques de type DOS (*Denial of service*).

Un autre avantage indirecte est que toutes les données saisies sont enregistrées dans une base interne à Google et peuvent être également copiées automatiquement dans

un tableau (sheet). Cela constitue un backup des informations entrantes dans le cas ou le SAV viendrait à être indisponible.

Design des formulaires

Comme la compagnie opère avec une marque blanche et des trois distributeurs, il fallait que le bandeau affiche une marques parmi les trois.
Dans chaque formulaire, il a fallu transporter le contexte sous la forme d’un champs de type texte.



Création Ticket

Enregistrement désactivé

*Obligatoire

Description du problème

Description rapide du problème *

Votre réponse

Description détaillée du problème

Si cela est possible, pouvez-vous décrire précisément les symptômes ? Indiquer le message d'erreur s'il y en a un. Si vous connaissez le nro de série ou le nro de commande ou le nro de facture, merci de nous les communiquer.

Votre réponse

Famille de produit *

Sélectionner

Formulaire de création de ticket auprès du Support

Formulaires

- Ouverture de ticket (trois variantes différentes selon le bandeau)
- Réouverture de ticket
- Demande de statut
- Demande de fermeture
- Demande de réouverture
- Réponse à une question du SAV

Chaque formulaire est accompagné d'un script écrit en langage Google Apps Script, qui effectue la collecte des champs remplis et qui effectue un appel de service REST synchrone auprès du SAV via ORDS (cf exemple en annexe). Ce même script notifie le client par mail du résultat de l'appel.
Il y a trois versions de formulaire et le script a été mutualisé dans une librairie.

Comme il n'y a pas d'authentification, les formulaires de demande de statut, par exemple, utilisent un contexte qui doit être transporté par le formulaire. Ce contexte est une chaine de caractère qui contient le code du ticket et une valeur de contrôle.

Contexte

Il est composé du code du ticket sur cinq caractères et d'un code de contrôle (hash code à partir d'informations variées). Il fallait proposer une valeur simple à retenir pour le client afin de faciliter sa communication avec le SAV, mais aussi décourager toute tentation de tester d'autres valeurs de tickets.

Comme le champs ne peut pas être en *Read Only*, rien n'empêche l'utilisateur de modifier le contenu de ce champs. Le risque se limite au fait que le script qui se

déclenche après la soumission du formulaire renverra un contexte que le service REST du SAV ne saura pas retrouver.

Conversations et Notifications

Les conversations et notifications s'appuient sur l'échange de mails. Il est envisageable d'avoir recours a des message SMS. J'ai testé le service OVH mais cela n'a pas été mis en production sur ce projet.

Les mails sont générés automatiquement par les scripts attachés aux formulaires Google à l'exception du cas où c'est un technicien qui décide d'envoyer une question.

L'analyste n'utilise donc que l'interface de backoffice pour effectuer son travail de tous les jours. Lorsqu'il doit demander des informations complémentaires pour instruire le traitement d'un ticket, il passe par une page standard de l'application SAV. La soumission de la page déclenche alors l'appel à un script PHP d'envoi de mail. (cf schéma d'architecture plus haut).

Lors du début du projet, la version *Oracle Free Tier* à la différence de sa version payante ne permettait pas l'envoi de mail , c'est à dire qu'il n'y avait pas moyen d'invoquer une passerelle SMTP. En conséquence, le contournement a consisté à se priver des fonctionnalités natives d'Oracle APEX et envoyer des mail via l'API REST de Google Gmail. Celle-ci implique une authentification avec signature des échanges mais je ne suis pas parvenu à utiliser les algorithmes de chiffrement sous PL/SQL pour déclencher OAUTH2 et il n'existe pas de package de connexion pour le PL/SQL proposé par Google. Les tentatives d'obtention d'un token valide (`https://oauth2.googleapis.com/token`) ont échoué avec la fonction de chiffrage `dbms_crypto.SIGN_SHA256_RSA`. et je me suis donc replié sur l'emploi d'une *Gateway* qui effectue l'authentification et l'envoi via du code PHP. Dans ce cas,

Google fournit une librairie PHP prête à l'emploi, *Google Client Library for PHP*. J'ai utilisé une instance *compute* disponible dans le tenant *free Tier* pour héberger cette passerelle et je l'ai munie d'un certificat SSL pour permettre la communication avec l'instance APEX.

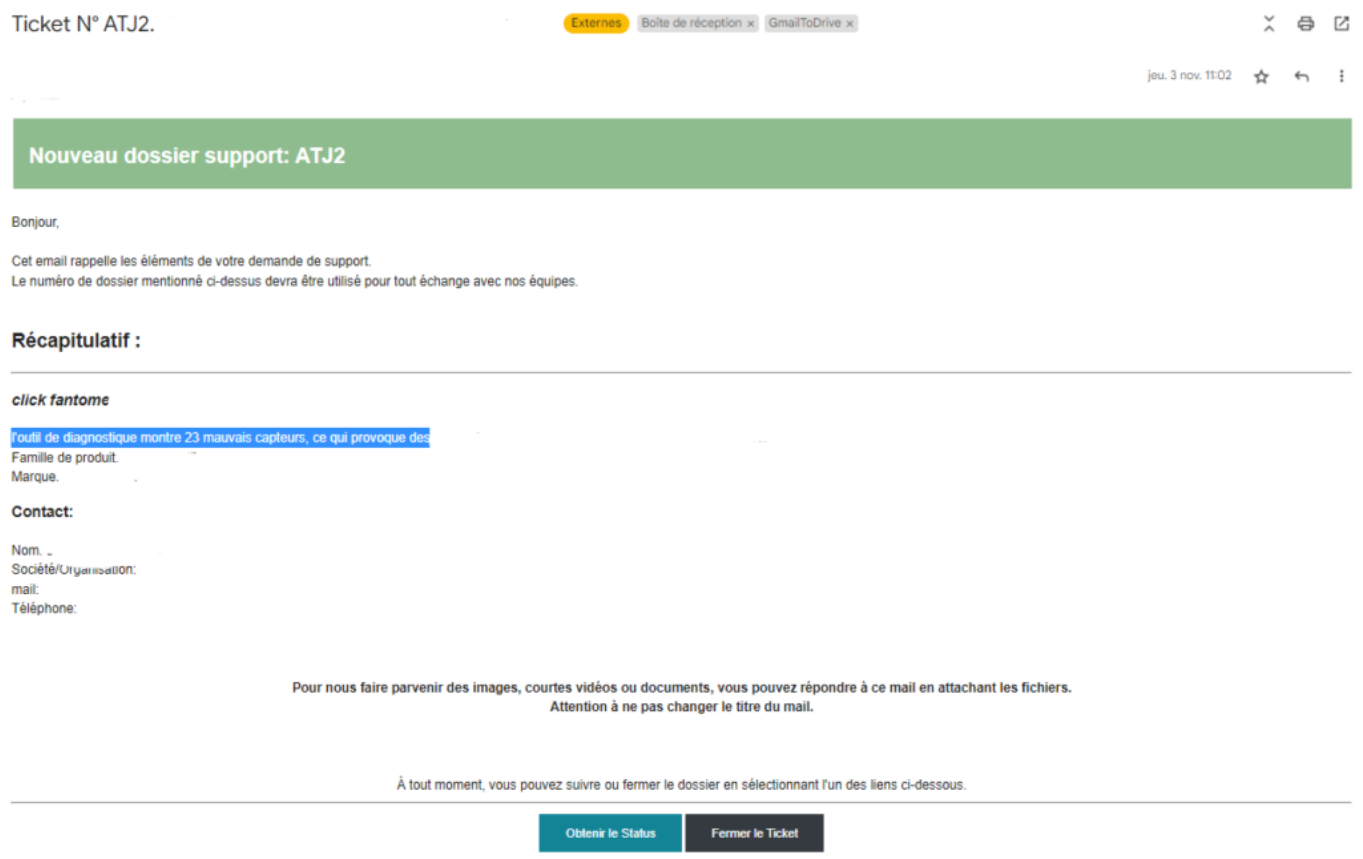
Actuellement, (décembre 2022), ces limitation ont été levées en partie puisqu'on peut désormais envoyer des mails avec certains seuils d'usage, mais qui auraient été acceptables dans le cadre du projet (cf annexes).

Types de mail

- Confirmation d'ouverture de ticket
- Demande de réponse à une question du SAV
- Fourniture du statut d'un ticket

Tous les mails contiennent dans leur objet, le contexte (Code Ticket et code de contrôle) et proposent dans leurs bas de page deux boutons permettant de demander, soit le statut du ticket soit sa fermeture, soit sa réouverture.

Donc, le client final n'a jamais l'initiative d'envoyer un mail au SAV. Il peut en revanche compléter son ticket avec des photos ou vidéos, par réponse au mail initial de notification d'ouverture d'un ticket.



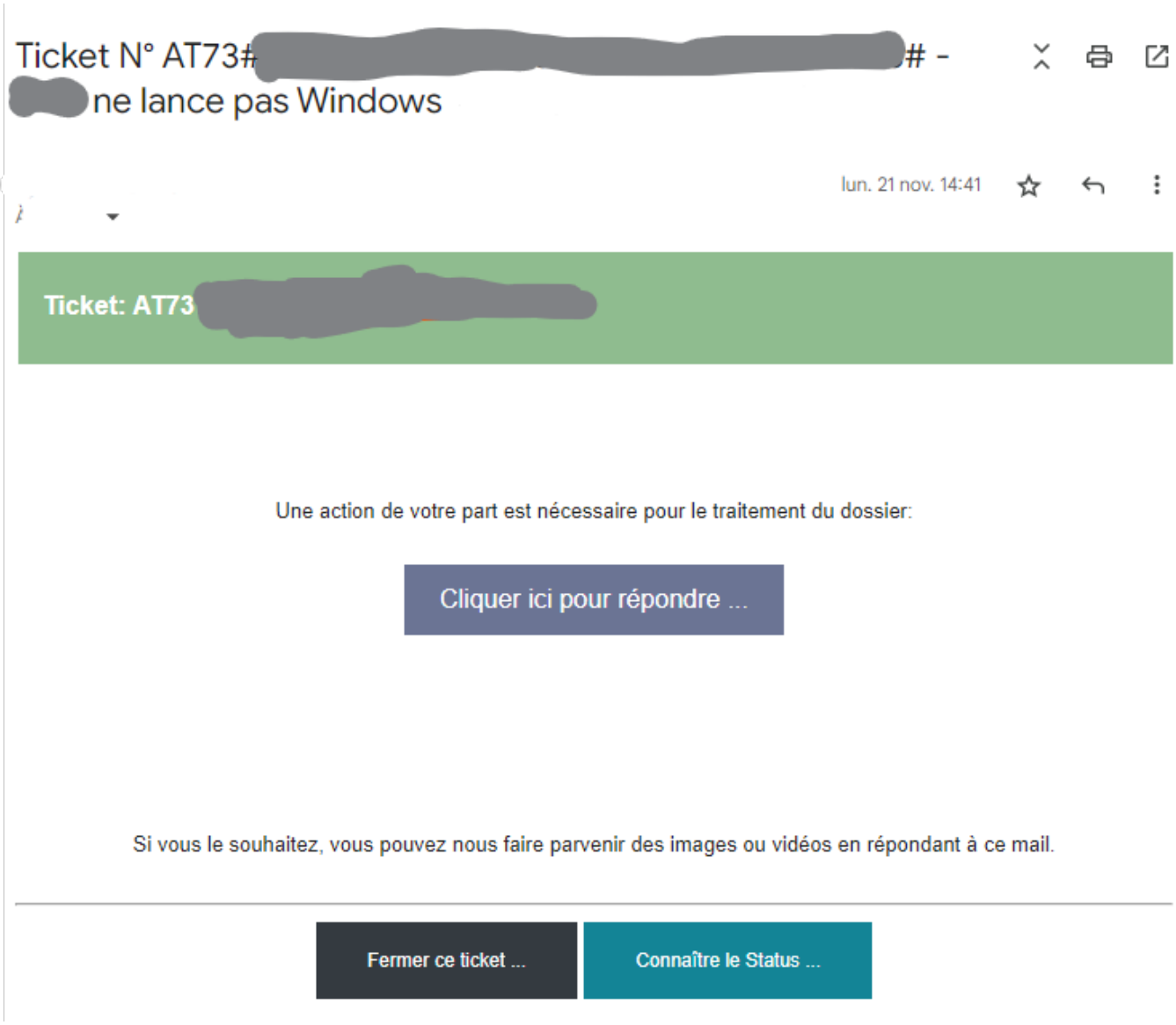
Mail récapitulant la création d'un ticket

Extraction des pièces attachées

Régulièrement, un script Google basé sur un *timer* se déclenche toutes les x minutes pour scruter les mails qui seraient munis de pièces attachées. Si c'est le cas, les fichiers sont recopiés dans un dossier de *Google Drive* nommé avec le nom du ticket et un appel REST informe le système qu'un pièce attachée est disponible (url de Google Drive). Ce mécanisme ne peut pas être synchrone (cette limitation est documentée par Google) et c'est la raison pour laquelle il faut effectuer du *poling*.

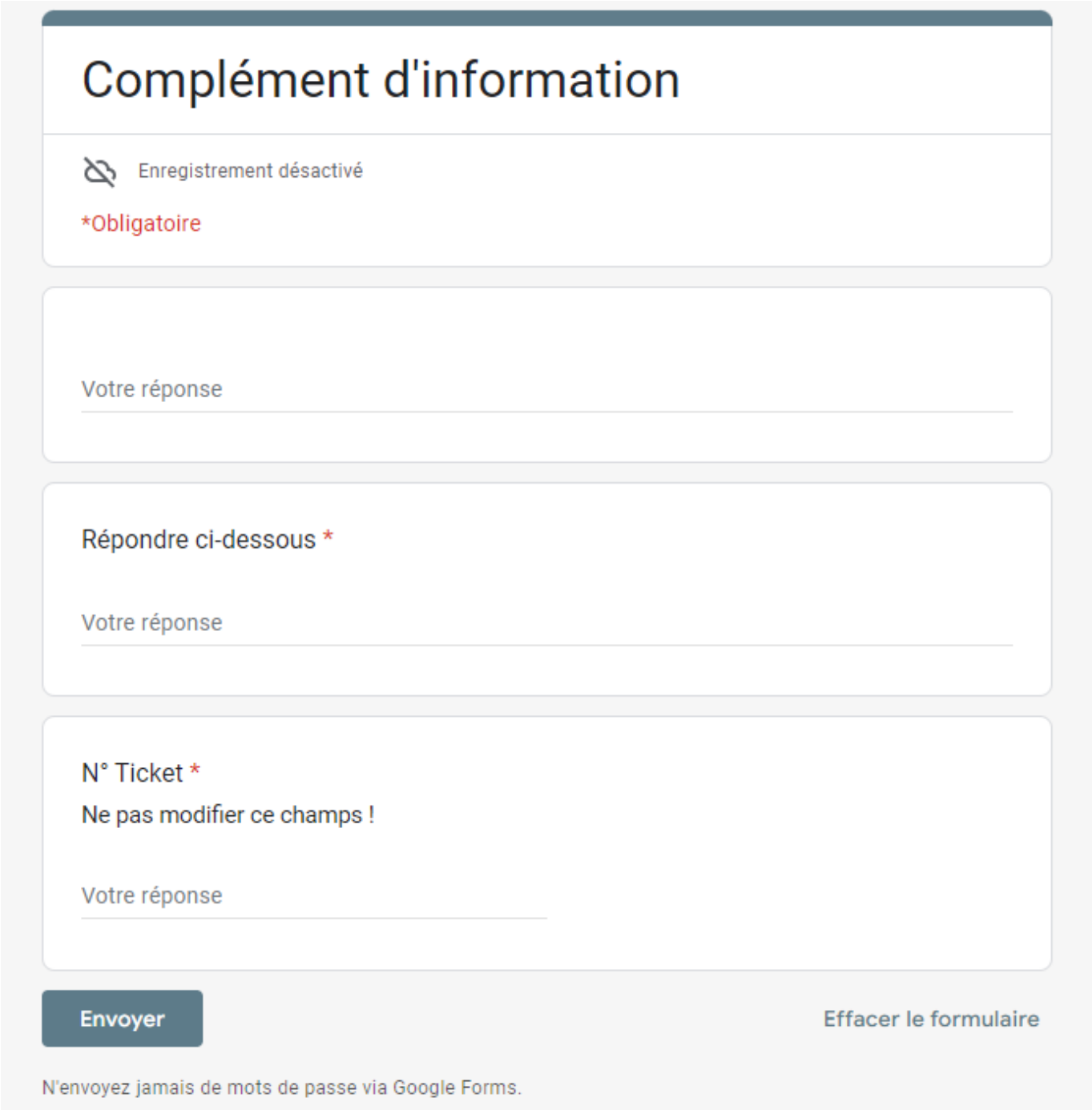
Alternative: Il aurait été possible d'utiliser un service d'intégration tel que Zapier ou Make (Integromat), mais cela aurait induit des couts d'usage.

Lorsque le technicien souhaite obtenir un complément d'information, il utilise l'application de backoffice pour rédiger la question et c'est le module de SAV qui effectue l'envoi de mail. Dans le corps du mail, il y a un texte générique et un bouton qui redirige vers le formulaire de réponse à une question du SAV.



Mail demandant des informations complémentaires

L'appui sur le bouton « *Cliquer ici pour répondre* » redirige vers un formulaire Google tel que celui-ci:



Formulaire pour saisir une réponse à une question du SAV

Lorsque le client veut accompagner sa demande d'ouverture de ticket par des photos ou vidéos, il doit les attacher en réponse au mail de confirmation qu'il aura reçu du

SAV.

Annexes

Exemple de script accompagnant un formulaire Google

Cet extrait de script Google illustre comment sont consommées les informations issues du formulaires, juste après sa soumission.
On voit ici un appel à un service REST de création de ticket dans SAV et l'envoi d'un mail de confirmation au client dans le cas où le statut de retour est ok.

Un simple mécanisme de clef de sécurité (API_KEY) permet une reconnaissance par le service REST qui est appelé.

```
function createTicket(e, canal) {
// ----------------------------------------
// Creates a new ticket in SAV subsystem
// and sends a mail to the customer
// Gpmfactory 2022
// ----------------------------------------
...
var url = '<SAV REST CREATE TICKET ENDPOINT>';
var formData = {
  'EMAIL': email,
  'NAME': t_name,
  'CUSTOMER': t_customer,
  'PHONE': t_phone,
  'DESCRIPTION': t_description,
  ...
  'API_KEY': 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
};

var options = {
  'method' : 'POST',
  'payload' : formData,
  'muteHttpExceptions': true
};
...
var response = UrlFetchApp.fetch(url, options);
var resp = response.getContentText();
rc = response.getResponseCode();
if (rc == "200" || rc == "200.0" ){
    MailApp.sendEmail(email, 'Ticket N° ' + resp, body, {'htmlBody' :
body });
    Logger.log('Mail envoyé');
  }
```

...

Limites d'Oracle Email Delivery Service pour Free Tier

A la date de décembre 2022
Source

Customers that sign up for a free Oracle Cloud trial are limited to:

- A volume of 200 emails per day where an email is defined as either a single recipient in the TO:, CC:, or BCC: fields, or a 2 MB chunk of data.Email examples:
    - A single request with 10 recipients (TO:, CC:, or BCC:) equals 10 emails.
    - A 10 MB email sent to a single recipient is equal to 10 MB divided by 2 MB per email. This equals 5 emails.
    - A single email request with a message size of 10 MB sent to 10 recipients is equal to 10 MB divided by 2 MB per email multiplied by 10 recipients. This equals 50 emails.
- 2,000 approved senders.
- Each user is limited to a maximum of two SMTP credentials.
- Sending rates are limited to 10 emails per minute.
- Inline attachments and external attachments.
- 2 MB maximum message size including base64 encoding and headers