

BACKELITE

Dashboard Performance

Documentation

Réalisé par Tommy Lopes le 09/01/2015

SOMMAIRE

I. Introduction

II. Description générale

1. Fonctionnalités
2. Personnalisation
3. Cache/historique
4. Données récupérées

III. Description technique

1. Schéma technique
2. Fonctionnement général
3. Technologies utilisées
4. Format de données

IV. Connecteurs

1. Google Analytics
2. App Figures
3. Urban Airship

V. Gestion des comptes et utilisateurs

VI. Base de données

VII. Evolutions

VIII. Installation

Dashboard Performance Documentation

I. Introduction

Le Dashboard Performance est une interface permettant de surveiller les indicateurs de Performance des services de nos clients. Il permettra de suivre les KPI définis avec nos clients en début de projet ou dans le cadre de l'offre performance Backelite.

Il existe de nombreuses solutions de dashboard (Geckoboard, Cyfe, Ducksboard, Leftronic) possédant de multiples connecteurs vers des services divers, notamment Google Analytics, Facebook, Salesforce, etc. Mais la majorité sont payants et non Open Source, et peu bénéficient de connecteurs destinés aux services de suivi de Performance: App Figures, Urban Airship, New Relic, Crashlytics, etc.

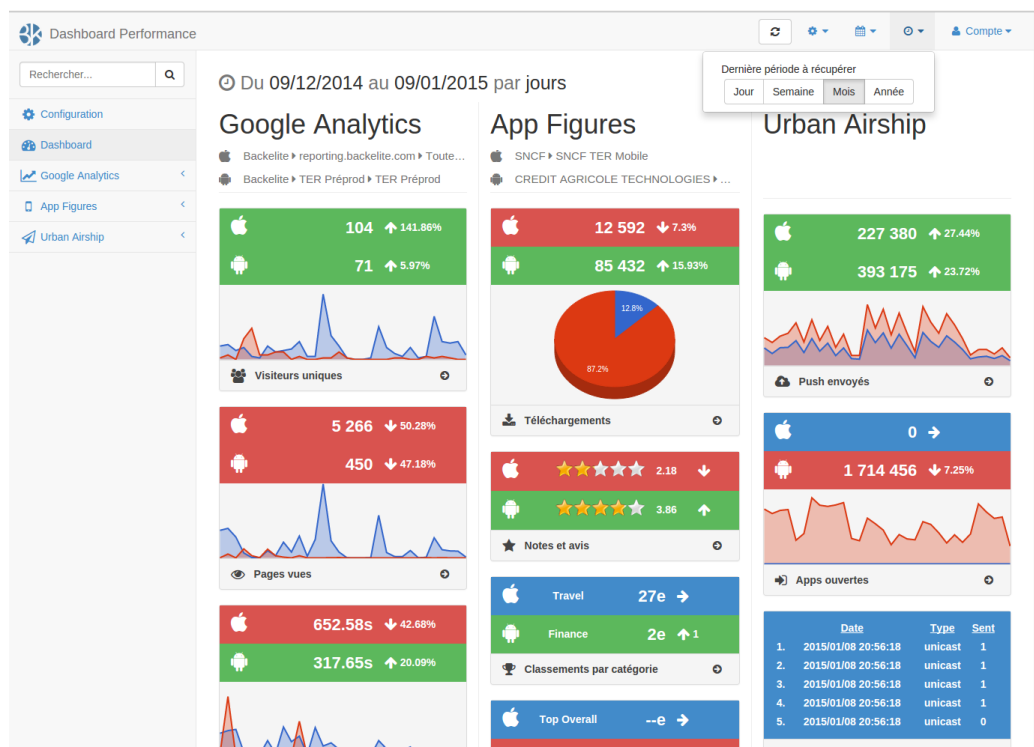
Nous avons décidé de développer notre propre solution qui permettra de se connecter avec des outils jugés indispensables pour notre besoin, et qui devra être relativement évolutive afin de se raccorder avec de nouveaux connecteurs en fonction des besoins clients futurs.

II. Description générale

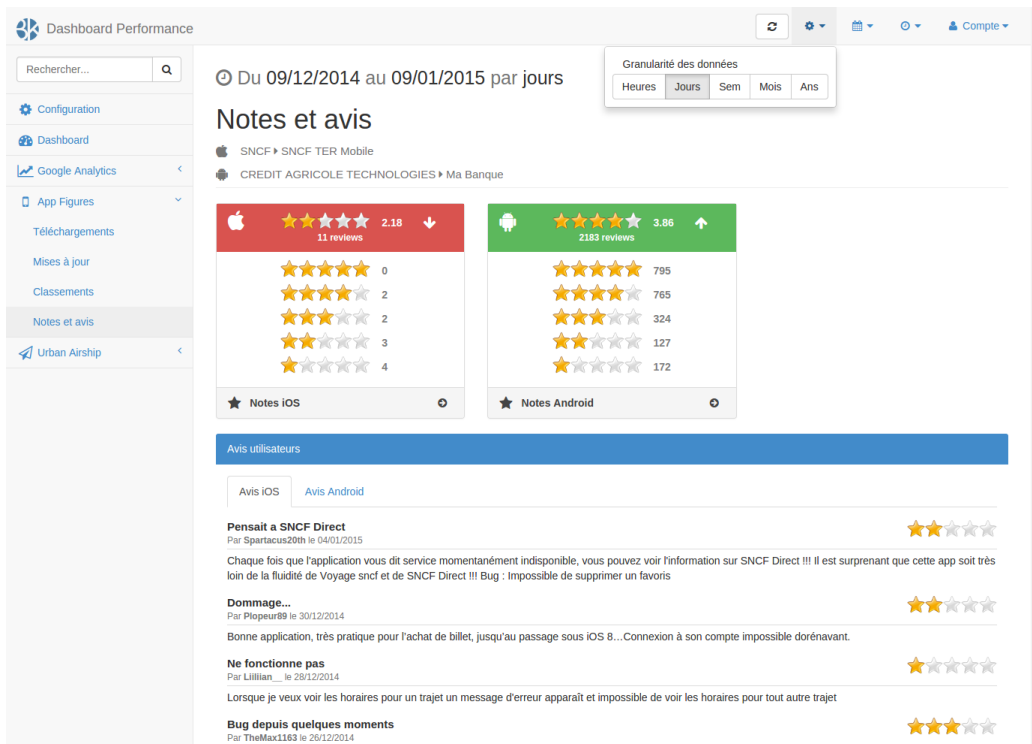
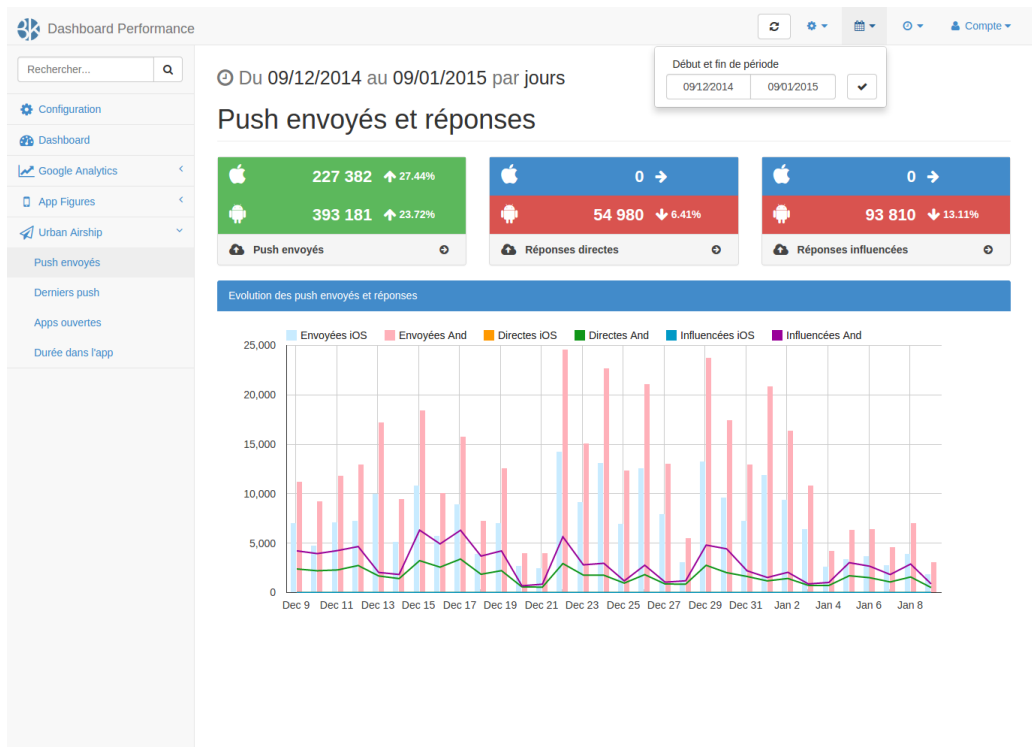
1) Fonctionnalités

Le projet ne faisant pour l'instant que l'objet d'un POC, il reste très basique dans sa conception et ses fonctionnalités, et sera amené à évoluer. Il agrège pour l'instant 3 connecteurs, Google Analytics, App Figures et Urban Airship. Il contient:

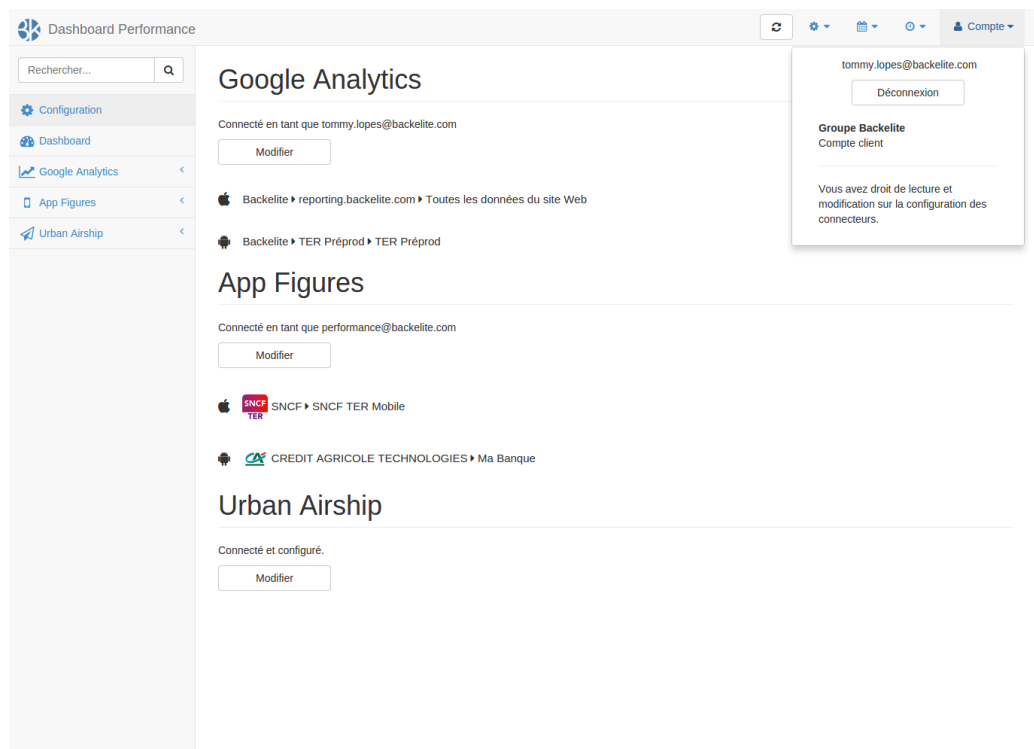
- un dashboard principal regroupant des informations essentielles, simples et concises, avec des indicateurs couleurs et l'évolution sur la période, fonctionnant réellement comme un tableau de bord et permettant au client de relever rapidement les points positifs et négatifs. Le but étant de donner une image synthétique et compréhensible en un coup d'œil de la situation de ses services.



- des pages détaillées donnant des informations complémentaires aux éléments du dashboard.



- une page de configuration des différents connecteurs.



2) Personnalisation

Les données peuvent être récupérées sur différentes périodes (dernier jour, semaine, mois, année ou choisir dans le calendrier) et peuvent être formatées selon différentes granularités (heure, jour, semaine, mois, année). Elles sont de plus comparées à la période précédente afin d'avoir une vue d'ensemble sur l'évolution des différents indicateurs.

Pour le moment, les plateformes prises en charge sont iOS et Android, on pourra ajouter WP par la suite.

Pour le moment l'interface est figée, aucune personnalisation n'est possible pour l'utilisateur (éléments déplaçables, configurables, multiples espaces de travail, ajout d'éléments etc.). L'interface est cependant totalement responsive (support Desktop/Tablette/Mobile).

3) Cache/historique

La problématique de stockage des données est compliquée à gérer (quantité énorme de données, fraîcheur des données etc.). Nous estimons que le dashboard ne devra la gérer que dans le cas où la source de données ne permet pas de requêter les historiques de données ou si l'API est trop lente pour atteindre un niveau d'expérience utilisateur suffisant. Le dashboard a pour vocation principale d'afficher les données de multiples sources et non de les télécharger.

A envisager:

- batcher les requêtes vers les services externes si les performances sont mauvaises et que la possibilité existe.
- établir un cache des requêtes des services tiers pour éviter de faire deux requêtes identiques.

Les données sont donc pour l'instant récupérées directement aux API des différents connecteurs, et les appels sont effectués à chaque affichage d'un indicateur. L'avantage est d'avoir des données toujours à jour, mais au moyen d'un nombre important de requêtes (par exemple la page de dashboard actuelle nécessite à elle seule plus de 50 requêtes externes), justifiées plus loin dans la description des API.

4) Données récupérées

Pour le moment les données récupérées sur le dashboard et les pages détaillées sont:

a) Google Analytics

- nombre de visiteurs uniques
- nombre de pages vues
- temps passé sur chaque page
- nombre de vues par nom de page
- durée moyenne de session
- taux de rebond

b) App Figures

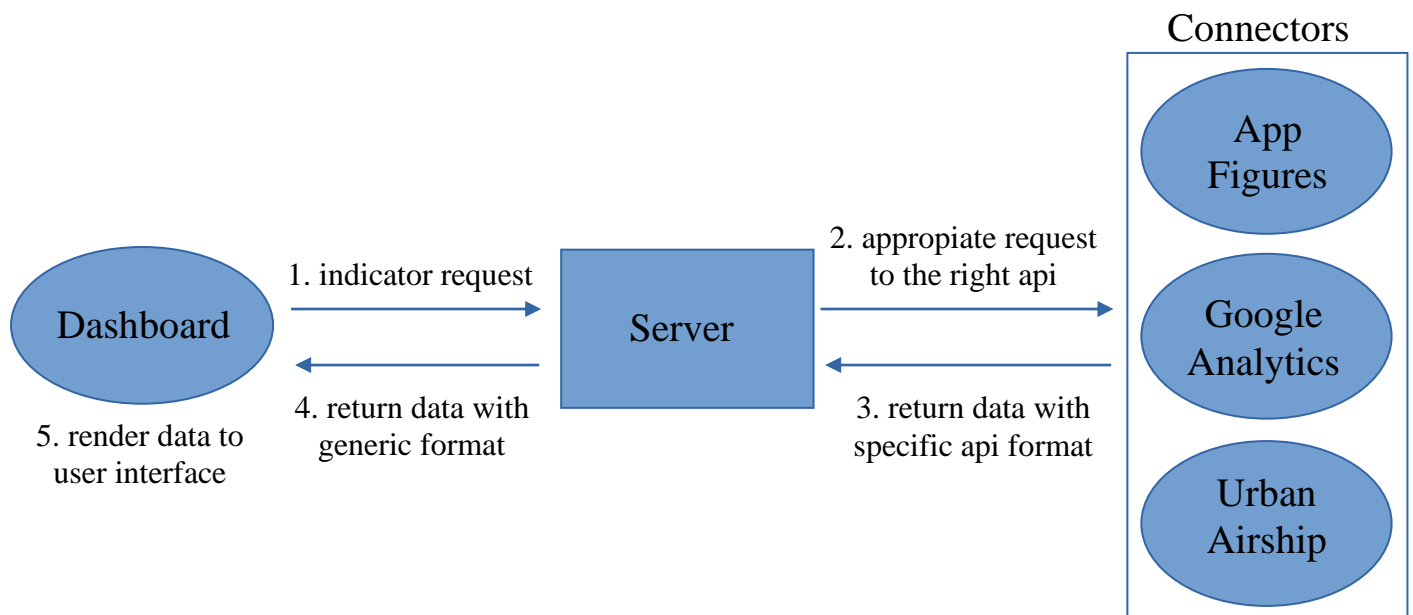
- nombre de téléchargements
- nombre de mises à jour
- revenus rapportés
- note de l'application
- commentaires utilisateurs
- classement par catégorie
- classement totalement

c) Urban Airship

- nombre de push envoyés
- nombre de réponses directes
- nombre de réponses influencées
- nombre d'apps ouvertes
- nombre d'apps opt-in ouvertes
- nombre d'apps opt-out ouvertes
- temps moyen passé sur l'app
- liste et description des derniers push envoyés

III. Description technique

1) Schéma technique



2) Fonctionnement général

Le projet est constitué de 3 grosses parties communiquant entre elles: le dashboard (front), le server (back) et les APIs externes (connecteurs).

De manière générale, lorsque le dashboard veut afficher un indicateur, le processus suit la chronologie suivante:

- le dashboard veut afficher un indicateur
- il envoie une requête au serveur avec le bon code indicateur et les options désirées
- le serveur reçoit la requête, la traite et définit le connecteur concerné
- il envoie alors la requête appropriée au connecteur concerné
- le connecteur traite la requête et envoie les données demandées avec son propre format
- le serveur reçoit les données, les traite et les réorganise
- il envoie alors les données demandées au dashboard dans un format générique
- le dashboard reçoit les données et les affiche à l'utilisateur

3) Technologies utilisées

a) Front Office

Les technologies utilisées pour le front sont AngularJS et bootstrap avec le template sb-admin-2 (<http://startbootstrap.com/template-overviews/sb-admin-2/>).

AngularJS est un framework JavaScript libre et open-source développé par Google, au même titre que MooTools, Prototype ou Dojo. Il a pour but de simplifier la syntaxe de JavaScript, et de combler les faiblesses de ce dernier en lui ajoutant de nouvelles fonctionnalités. Ceci permet notamment de faciliter la réalisation d'applications web monopages.

AngularJS nous permet entre autre ici de gérer très facilement et efficacement le routage de notre application et de faire du monopage, et donc ne charger que le contenu et les vues qui nous intéressent à chaque changement de route. Il est également très efficace dans sa conception car il permet de faire du MVC facilement, et ainsi updater les vues HTML automatiquement lors de la modifications des données, par l'intermédiaire de contrôleurs, et de manière transparente. Plus d'informations sur son utilité et fonctionnement: <https://angularjs.org/>.

Plusieurs plugins javascript sont utilisés pour afficher graphiquement les données à l'utilisateur:

Google Charts offre énormément de possibilités afin de visualiser les données sous formes de graphiques interactifs, totalement réalisés en HTML5/SVG, personnalisables, responsive et simple d'utilisation. Ils sont notamment utilisés pour les graphiques courbe, aire, camembert, géographique, gauge, et les graphiques combinés.

Plus d'informations sur son fonctionnement et possibilités:

<https://developers.google.com/chart/interactive/docs/gallery>

RateIt est un plugin jQuery afin de visualiser des données sous forme d'étoiles, avec de nombreuses options et personnalisations. Ici utile pour visualiser les notes des applications sous forme d'étoiles.

<http://rateit.codeplex.com/>

DataTables est un plugin jQuery et est utilisé pour visualiser les données sous forme de tableaux. Très flexible et facile d'utilisation, il offre de nombreuses possibilités d'interaction avec l'utilisateur, directement intégrées au tableau (tris, filtres, recherche, pagination etc.).

<https://datatables.net/>

b) Back Office

La technologie utilisée pour le back est intégralement Node.js avec le framework Express.

Node.js est une plateforme logicielle libre et événementielle en JavaScript orientée vers les applications réseau qui doivent pouvoir monter en charge. Elle utilise la machine virtuelle V8 et implémente sous licence MIT les spécifications CommonJS. Node.js contient une bibliothèque de serveur HTTP intégrée, ce qui rend possible de faire tourner un serveur web sans avoir besoin d'un logiciel externe comme Apache ou Lighttpd, et permettant de mieux contrôler la façon dont le serveur web fonctionne.

<http://nodejs.org/>

Express est un framework nodejs très simple et flexible, qui fournit un ensemble robuste de fonctionnalités pour faciliter le développement d'applications web.

Il est utile ici entre autre pour faciliter le routing, la gestion des sessions et pour ses fonctions de template (Embedded JavaScript ejs).

<http://expressjs.com/>

4) Format des données

Afin d'afficher un indicateur, le dashboard envoie une requête au serveur pour les données.

La requête aura la forme suivante:

REQUEST

GET

`http://localhost:1337/get/api/data/?codeName={codeName}&start={start}&end={end}&granularity={granularity}&past={past}`

ARGUMENTS

- `codeName`: code de l'indicateur, par exemple 'uniqueVisitors'. Valeurs possibles: uniqueVisitors, uniqueVisitorsCountry, pageViews, pageViewsPath, avgTimePage, avgSessionDuration, bounceRate, operatingSystemSessions, allTimeDownloads, downloads, allTimeRevenue, revenue, allTimeUpdates, updates, ranks, ratings, sends, opens, timeinapp, optins, optouts, direct, influenced, listing
- `start`: la date de départ de l'intervalle de données à renvoyer. Format 'YYYY-MM-DD'
- `end`: la date de fin de l'intervalle de données à renvoyer. Format 'YYYY-MM-DD'
- `granularity`: optionnel. A utiliser pour avoir une granularité dans les données renvoyées. Par exemple avec 'daily' les données renvoyées seront détaillées par jour sur l'intervalle demandé. Valeurs possibles: hourly, daily, weekly, monthly
- `past`: optionnel. 'true' pour avoir la comparaison avec la période précédente (sans granularité)

SUCCESS RESPONSE OBJECT

- `codeName`: code de l'indicateur retourné
- `code`: état de la réponse. 'OK'
- `status`: code http de la réponse (ex: 200, 404, 500 etc.)
- `ios`: objet contenant les données ios
- `android`: objet contenant les données android

Les objets ios et android sont de la forme:

- `dates`: tableau de dates au format 'YYYYMMDDHHmmss'
ex: ['20141124000000', '20141224000000', ...]
- `datas`: tableau de tableaux de données pour chaque date
ex: [[42], [1337], ...] ou [['foo', 'bar', 42], ['toto', 'tutu', 1337], ...]

ERROR RESPONSE OBJECT

- `code`: état de la réponse. 'FAIL'
- `status`: code http de la réponse (ex: 200, 404, 500 etc.)
- `message`: message d'erreur personnalisé

Si les données demandées sont générales et ne nécessitent pas la comparaison entre iOS et android alors l'objet renvoyé aura directement les champs 'dates' et 'datas' au lieu de 'ios' et 'android'

- codeName
- code
- status
- dates
- datas

Si l'option 'past' est égale à true, l'objet renvoyé aura la forme d'une comparaison avec les données passées et les données présentes, sans granularité. Le champs 'dates' sera donc un tableau vide et le champs 'datas' sera un tableau de la forme suivante

[['past', datasPast], ['now', datasNow]] ex: [['past', 42], ['now', 1337]]

IV. Connecteurs

1) Google Analytics

a) Description

Google Analytics est un service gratuit d'analyse d'audience d'un site Web ou d'applications utilisé par plus de 10 millions de sites. L'outil représente plus de 80% du marché mondial. C'est ainsi le service d'analyse de visites de sites web le plus utilisé au monde.

Il permet entre autres d'avoir des informations sur les utilisateurs, les sessions, les pages consultées, les écrans vus dans une app, les recherches, les événements, les performances, et bien d'autres.

Plus de features ici: <http://www.google.com/analytics/features/>

L'API Core Reporting va nous permettre de récupérer toutes ces données avec plusieurs indicateurs en même temps, en jouant sur les metrics, les dimensions, les filtres, les tris, les périodes etc. Elle est très performante et offre de grandes possibilités de personnalisation des indicateurs surveillés. Mais on ne peut requêter qu'un compte à la fois.

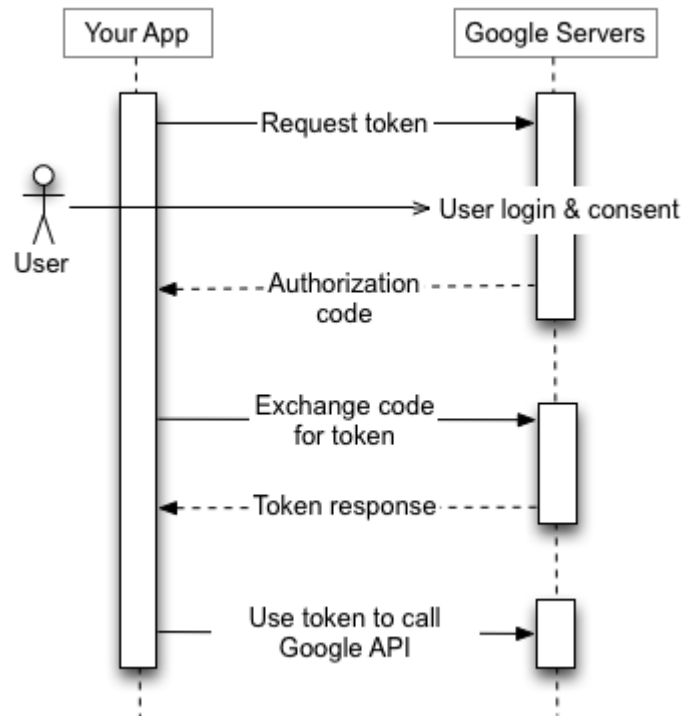
Doc de l'API: <https://developers.google.com/analytics/devguides/reporting/core/v3/>

L'API Management va nous permettre de récupérer toutes les informations nécessaires sur les comptes analytics disponibles sur un compte google. Elle permet de parcourir tous les comptes classés par 'Account', 'Property' et 'Profile', d'en récupérer les noms, ids, etc. Les ids sont nécessaires pour requêter l'API Core Reporting et donc récupérer les données d'un compte spécifique.

Doc de l'API: <https://developers.google.com/analytics/devguides/config/mgmt/v3/>

b) Authentication

L'authentification à un compte google se fait via la procédure OAuth2 uniquement, une évolution assez récente de OAuth1.0a, plus simple dans son implémentation et son utilisation, et plus sécurisée. Elle est basée sur l'échange de tokens de courte durée de vie pour s'identifier.



La procédure passe donc par plusieurs étapes, dans l'ordre:

- l'application demande un 'request token' au serveur google, en donnant le client_key et les scopes désirés
- celui-ci répond avec le lien d'une page où l'utilisateur va pouvoir se logger en toute sécurité avec un écran de consentement concernant tous les scopes demandés par l'application (accès au mail, infos publiques, données analytics etc.)
- au succès de la connexion, le serveur google envoie un code d'autorisation à l'application à l'adresse de redirection spécifiée lors de la première requête
- l'application récupère le code d'autorisation et l'utilise pour demander un 'access token' au serveur google
- si le code d'autorisation est bon et que tout se passe bien, le serveur google répond avec un 'access token' qui sera utilisé pour les appels aux APIs google, notamment pour récupérer les données analytics de l'utilisateur loggé

L'access token récupéré n'a qu'une durée de vie d'une heure en moyenne, après quoi les appels aux APIs seront refusés. Afin de palier à ce problème, il est possible de préciser le type d'accès désiré lors de la demande de token, online ou offline. Si on précise que l'on désire un accès offline, l'utilisateur est prévenu sur l'écran de consentement et le serveur google nous envoie un 'refresh token' avec l'access token, qui permettra de redemander un nouvel access token valide lorsque l'actuel expire. Le refresh token a une durée de vie illimitée, et le seul moyen de l'annuler est une action directe et manuelle de l'utilisateur dans la console de développeur google.

L'access token et le refresh token sont tous les deux stockés en base de données pour les récupérer à chaque visite de l'utilisateur. Le serveur de l'application s'occupe automatiquement de demander un nouvel access token valide toutes les 45min et de le stocker.

Plus d'infos sur la procédure OAuth2 et ses options:

<https://developers.google.com/accounts/docs/OAuth2WebServer>

c) Quota limite

La limite de requêtes pour les APIs google est de 50 000 par application par jour (peut être augmenté en contactant google) et de 10 requête par seconde par IP (non négociable).

<https://developers.google.com/analytics/devguides/reporting/core/v2/limits-quotas>

La limite des 10 requête par seconde est un vrai problème, étant donné que l'affichage du dashboard à lui seul nécessite une 20aine de requête toutes lancées en asynchrone.

Pour cela on utilise une queue qui limite les requêtes google à 10 par seconde et attend que les autres puissent être lancées.

Le module nodejs utilisé est 'node-rate-limiter' qui permet de faire ça très simplement.

<https://github.com/jhurliman/node-rate-limiter>

Un autre problème se pose: l'application réalise toutes les requête google pour tous les utilisateurs depuis le serveur et donc avec la même IP. Il suffit que plusieurs utilisateurs veuillent afficher le dashboard en même temps et les performances vont considérablement chuter.

Pour ce cas précis, où une application requête les apis google à la place de ses utilisateurs, google introduit 2 paramètres pour les requêtes: userIp et quotaUser.

On utilise ici quotaUser pour requêter les apis google, afin d'identifier les utilisateurs finaux et d'appliquer un quota unique à chacun d'eux.

<https://developers.google.com/analytics/devguides/reporting/realtime/v3/parameters>

d) Retours

L'API google analytics ne permet pas de requêter plusieurs comptes en même temps. Afin de comparer les données iOS et Android il faut donc requêter les deux comptes séparément, augmentant considérablement le nombre de requêtes.

L'affichage d'un simple indicateur google du dashboard demandera alors à lui seul 6 requêtes:

le total sur la période actuelle, le total sur la période précédente et le détail avec granularité pour le graphique, et ce 2 fois (iOS + Android).

2) App Figures

a) Description

App Figures est une plate-forme de reporting pour les développeurs mobiles qui récupère et visualise automatiquement les données iOS et Android de l'App Store et du Play Store. Il permet entre autres d'avoir des informations sur les ventes, les avis utilisateurs, les notes, les classements, les revenus, les ads etc.

L'API permet de récupérer toutes ces données, pour plusieurs comptes en même temps, différentes périodes, différentes granularités, plusieurs méthodes de groupement (produit, date, pays etc.), mais qu'un indicateur à la fois.

Doc de l'API: <http://docs.appfigures.com/>

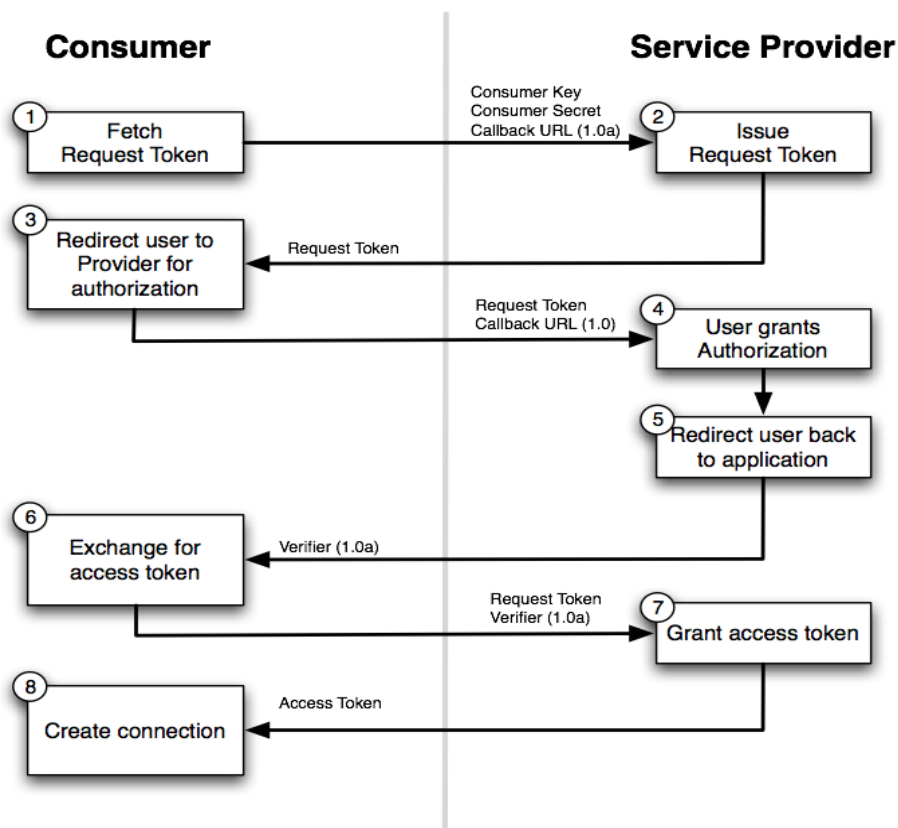
b) Authentication

L'authentification à un compte app figures se fait via une authentification basique (http auth basic) ou via la procédure OAuth1.0a. Dans les deux cas l'authentification nécessite les clés publique et secrète de l'application obtenues depuis un compte appfigures (client_key et client_secret).

L'authentification basique se fait avec le mail et le mot de passe de l'utilisateur et n'est possible que dans le cas où ces derniers correspondent au compte avec lequel a été obtenu le client_key. On ne l'utilisera donc que dans le cas où le client n'a pas de compte propre et qu'on utilisera le compte de backelite, correspondant au client_key de l'application.

Dans le cas où le client possède son propre compte appfigures, on utilisera OAuth1.0a pour l'authentifier et requêter à sa place.

Comme pour OAuth2, cette procédure est basée sur l'échange de tokens de courte durée de vie, mais l'access token final utilisé pour les appels à l'API est de durée illimitée.



La procédure passe donc par plusieurs étapes, dans l'ordre:

- l'application demande un 'request token' au serveur appfigures, en donnant le client_key, le client_secret et les scopes désirés
- celui-ci répond avec un token et un token_secret
- l'utilisateur est alors dirigé sur la page d'autorisation créée avec les tokens, où il va pouvoir se logger en toute sécurité avec un écran de consentement concernant tous les scopes demandés par l'application (publique, privé, produits etc.)
- au succès de la connexion, le serveur appfigures envoie un code de vérification à l'application à l'adresse de redirection spécifiée lors de la première requête
- l'application récupère le code de vérification et l'utilise pour demander un 'access token' au serveur appfigures
- si le code de vérification est bon et que tout se passe bien, le serveur appfigures répond avec un 'access_token' et un 'token_secret' qui seront utilisés pour les appels à l'API, et donc pour récupérer les données des produits de l'utilisateur loggé

L'access token et le secret token ont une durée de vie illimitée et sont stockés en base de données pour les récupérer à chaque visite de l'utilisateur.

<http://docs.appfigures.com/api/reference/v2/authentication>

c) Quota limite

La limite de requêtes pour l'API appfigures est de 1000 par compte par jour (peut être augmenté en les contactant).

En admettant que plusieurs utilisateurs pourraient utiliser le compte d'une entreprise, 1000 requêtes par jour par compte ne suffit pas du tout et il faudra penser à une solution.

D'autant plus vrais si certains clients utilisent le compte Backelite.

<http://docs.appfigures.com/api/rate-limits>

d) Retours

La granularité acceptée par l'API change selon les indicateurs. Pour certains les granularités acceptées sont uniquement l'heure et le jour, pour d'autres le jour, la semaine et le mois.

Google acceptant toutes les granularités citées, le dashboard s'aligne dessus et les accepte toutes aussi.

N'ayant pas eu le temps de traiter cette problématique, pour l'instant dans le cas où l'API appfigures n'accepte pas une granularité choisie, on prendra la granularité acceptée la plus proche.

3) Urban Airship

a) Description

Urban Airship plate-forme d'outils push et reporting pour les technologies mobiles. Il fournit des outils et des services conçus pour les développeurs d'applications mobiles pour améliorer facilement leurs applications avec des services en engagement d'audience, tels que des notifications push, des riches notifications texte interactives et des pages in-app.

Il permet entre autres d'avoir des informations sur les pushes envoyés, les réponses aux notifications, les apps ouvertes et d'autres statistiques.

L'API permet de récupérer ces données et intègre automatiquement la comparaison entre iOS et Android à ses réponses. Les personnalisations de réponses de l'API sont toutefois limitées et ne propose que de la granularité, pas de filtre, ni tri, ni groupement ou autre.

Doc de l'API: <http://docs.urbanairship.com/api/ua.html>

b) Authentification

L'authentification à un compte urban airship se fait uniquement via une authentification basique en donnant l'App Key et l'App Master Secret. Ces deux clés sont fournies dans l'interface utilisateur d'une application d'un compte urban airship: menu latéral gauche > settings > API keys.

Ces deux clés sont stockées en base de données pour les récupérer à chaque visite de l'utilisateur.

c) Quota limite

Aucune précision n'est donnée par urban airship quand à la limite du nombre de requêtes par compte, IP ou utilisateur.

Il n'y a cependant jamais eu de problème en réalisant beaucoup de requête à la seconde et par jour.

A voir pour se renseigner directement auprès d'urban airship.

d) Retours

Pour la plupart des indicateurs, l'API retourne 100 résultats maximum avec une URL donnant sur la page de résultats suivante. Pour les demandes de données sur de longs intervalles ou avec une faible granularité qui généreront plus de 100 résultats, il est donc obligé de requêter en boucle sur les pages suivantes pour récupérer tous les résultats voulus. Ce qui dégrade parfois considérablement les performances.

De plus l'API demande obligatoirement une granularité, et ne donne donc pas le résultat total d'un indicateur quand c'est voulu. Par exemple si on veut le nombre total de push envoyés sur une certaine période, il faut obligatoirement demander une granularité sur le jour, recevoir les réponses sur tous les intervalles et tout additionner, peu pratique.

L'API n'accepte pas les granularités sur la semaine et l'année.

N'ayant pas eu le temps de traiter cette problématique, pour l'instant dans le cas où l'API urban airship n'accepte pas une granularité choisie, on prendra la granularité acceptée la plus proche.

L'API ne permet pas de récupérer les informations de l'application requêtée (nom, icône, etc.).

V. Gestion des comptes et utilisateurs

Le projet faisait pour l'instant l'objet d'un POC, la gestion des utilisateurs est basique. Il existe 3 types d'utilisateurs

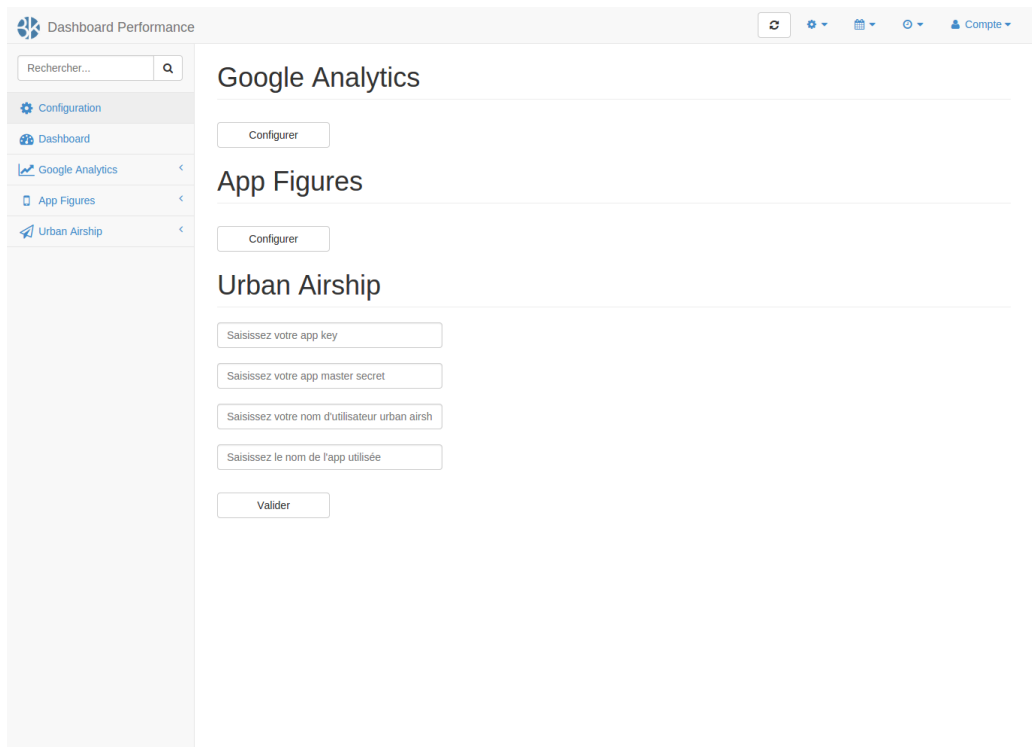
- Admin: son interface lui permet de créer des comptes client et des utilisateurs et d'en avoir une vue d'ensemble
- Client: le client est rattaché à un compte client, a accès au dashboard et autres données analytiques, et a un droit de modification sur la page de configuration des connecteurs
- User: le user est rattaché à un compte client, a accès au dashboard et autres données analytiques, mais a uniquement un droit de lecture sur la page de configuration des connecteurs

Chaque utilisateur (hormis les admin) est rattaché à un compte client dans lequel les connecteurs sont à configurer. Un des utilisateurs client du compte devra alors choisir les produits iOS et Android pour chaque connecteur qui seront utilisés pour le dashboard.

Pour google analytics, le client doit s'authentifier au compte google voulu puis choisir un produit iOS et un produit Android, et valider. Les données d'authentification et les Ids des 2 produits seront alors stockés en base de données et actifs pour tous les utilisateurs rattachés au compte client.

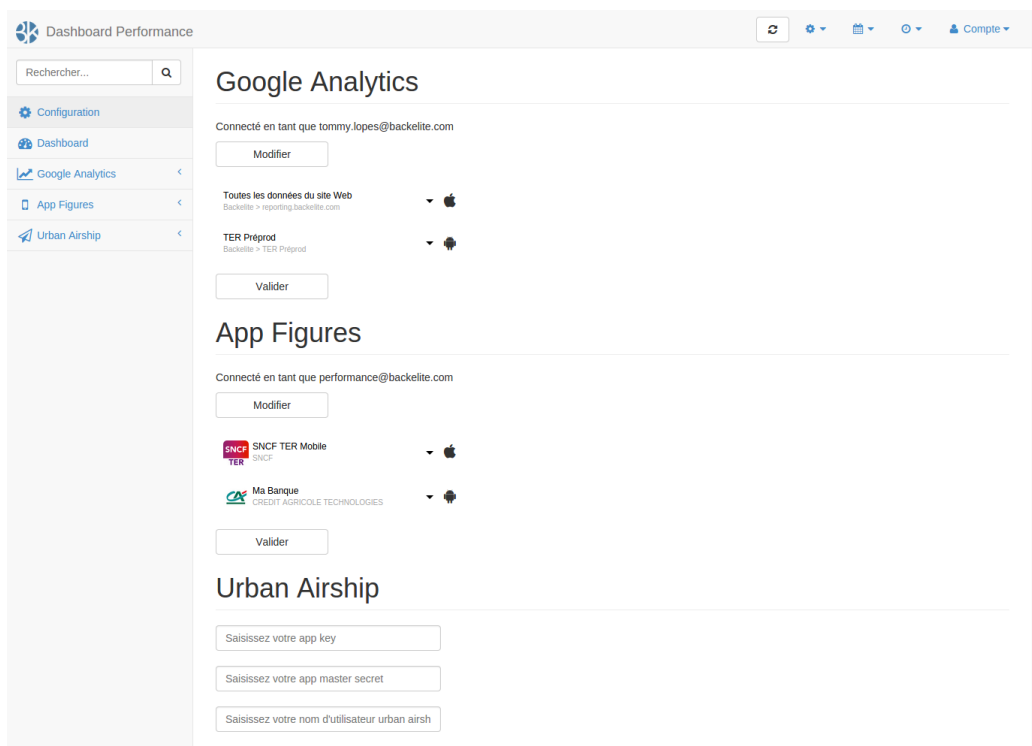
De même pour appfigures, uniquement si l'entreprise possède son propre compte. Dans le cas contraire le compte appfigures backelite sera utilisé et les 2 produits seront validés directement par l'admin lors de la création du compte client.

Pour urban airship, le client doit saisir l'App Key et l'App Master Secret de l'application à utiliser. Ces deux clés seront alors stockées en base de données et actifs pour tous les utilisateurs rattachés au compte client.



En cliquant sur connexion, une fenêtre d'authentification s'ouvre.

Après connexion, il faut choisir les produits dans les liste déroulantes des connecteurs et cliquer sur valider pour sauvegarder les choix. Un message informe du succès ou de l'échec de l'opération.



L'interface administrateur est composée de plusieurs écrans de gestion des comptes.

- Une page de visualisation des comptes et utilisateurs et leurs infos sous forme de tableaux

Dashboard Performance Compte

Rechercher...

Comptes et utilisateurs

+ Ajouter un compte

+ Ajouter un utilisateur

Comptes client et utilisateurs

Comptes client

Élément par page : 10 Rechercher :

Nom	Google Analytics	Compte appfigures	App Figures	Urban Airship
ADMIN	attente	backelite	attente	attente
Backelite	configuré	client	configuré	configuré
Epita	configuré	backelite	attente	configuré
Test	attente	client	attente	attente
Test2	attente	client	attente	attente
Test3	attente	backelite	attente	attente
Test4	attente	client	attente	attente

1 à 7 sur 7 éléments

Précédent 1 Suivant

Utilisateurs

Élément par page : 10 Rechercher :

Email	Type	Compte
client@backelite.com	client	Backelite
perfdashboard@backelite.com	admin	ADMIN
tommy.lopes@backelite.com	client	Backelite
tommy.lopes@epita.fr	client	Epita
tommyblopes@msn.com	client	Test
user2@backelite.com	user	Backelite
user@backelite.com	user	Backelite

1 à 7 sur 7 éléments

Précédent 1 Suivant

- Une page afin d'ajouter un compte client
- Saisit du nom de compte et si oui ou non le client possède son propre compte App Figures (si non, le compte Backelite est utilisé et l'admin doit configurer l'app iOS et l'app Android dont les utilisateurs associés à ce compte auront accès)

Dashboard Performance Compte

Rechercher...

Comptes et utilisateurs

+ Ajouter un compte

+ Ajouter un utilisateur


Ajouter un compte client


Nom du compte client

Compte App Figures propre au client?

Non

Configuration avec le compte Backelite

Ma Banque CREDIT AGRICOLE TECHNOLOGIES 

Ma Banque CREDIT AGRICOLE TECHNOLOGIES 

Ajouter

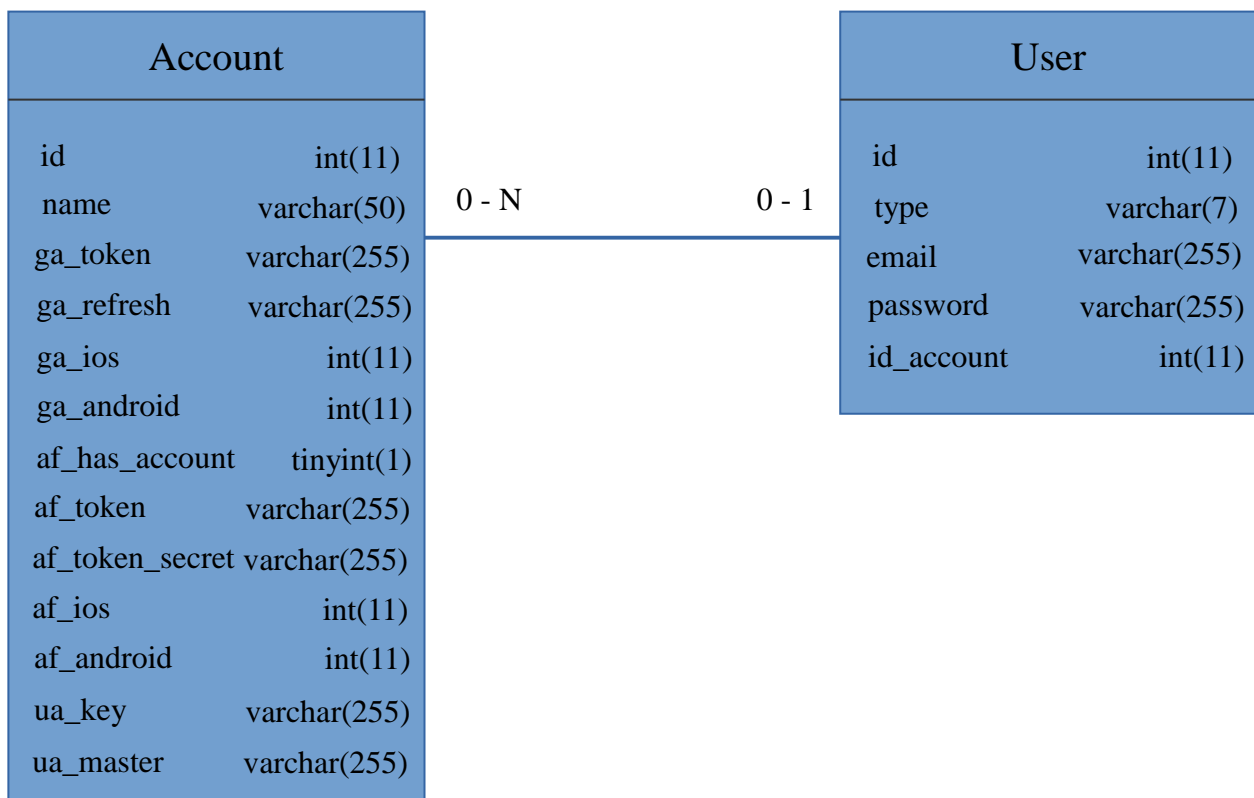
- Une page afin d'ajouter un utilisateur

Saisie de l'email, du mot de passe, du type utilisateur (utilisateur basique, client – ayant droit de modification de la configuration des connecteurs, ou admin) et du compte client auquel il sera associé

The screenshot shows a web interface titled "Ajouter un utilisateur" within a "Dashboard Performance" context. On the left, a sidebar contains a search bar and navigation links: "Comptes et utilisateurs", "Ajouter un compte", and "Ajouter un utilisateur". The main form area includes the following elements:

- A search bar with the placeholder "Rechercher..." and a magnifying glass icon.
- A title "Ajouter un utilisateur".
- An input field labeled "Email utilisateur".
- A label "Saisissez un mot de passe" followed by a password input field.
- A label "Saisissez à nouveau le mot de passe" followed by a confirmation password input field.
- A label "Choisissez le type de l'utilisateur" followed by a dropdown menu currently showing "Utilisateur".
- A label "Choisissez le compte client associé (configuration des connecteurs)" followed by a dropdown menu currently showing "Backelite".
- An "Ajouter" button at the bottom of the form.

VI. Base de données



Account

- name: nom du compte client (nom de l'entreprise ou autre)
- ga_token: access token du compte google analytics
- ga_refresh: refresh token du compte google analytics
- ga_ios: id du profile correspondant au produit iOS
- ga_android: id du profile correspondant au produit Android
- af_has_account: 1 si le client a son propre compte appfigures, 0 sinon
- af_token: access token du compte appfigures
- af_token_secret: token secret du compte appfigures
- af_ios: id du produit iOS
- af_android: id du produit Android
- ua_key: App Key de l'application urban airship
- ua_master: App Master Secret de l'application urban airship

User

- type: admin, client ou user

Un user peut n'appartenir à aucun compte seulement s'il est admin, les autres sont obligatoirement rattachés à un compte.

Un compte peut avoir 0 ou plusieurs users.

VII. Evolutions

- Mettre en place le server HTTPS (fait simplement en quelques lignes de code, il suffit simplement d'un certificat SSL valide et reconnu)

<http://stackoverflow.com/questions/11744975/enabling-https-on-express-js>

- Mettre en place un chiffrement symétrique pour un stockage sécurisé des tokens et app keys et différents connecteurs en base de données. Ne surtout pas les laisser en dur comme c'est le cas actuellement.

- Résoudre le problème de la limite des 1000 requêtes par jour pour App Figures. Pour le moment, au dépassement de la limite les requêtes ne passent plus et les indicateurs app figures affichent alors une erreur sur le dashboard.

- Résoudre le problème de la limite des 10 requêtes par seconde pour Google Analytics. Comme dit précédemment, une queue pour transiter les requêtes afin de ne pas dépasser la limite a été mis en place, mais une ou deux requêtes échouent de temps en temps, aléatoirement. Je n'ai pas eu le temps d'identifier le problème. Problème d'implémentation de la queue ? De l'API google ?

- Améliorer les performances générales

VIII. Installation

1) Installation de Node.js et npm

```
$ sudo apt-get install nodejs-legacy npm
```

2) Installation de MySQL

```
$ sudo apt-get install mysql-server
```

3) Récupération du projet

```
$ svn checkout http://subversion.backelite.com/backelite/bkperfdashboard/
```

4) Aller à la racine du projet

```
$ cd bkperfdashboard/trunk
```

5) Importer la base de données

```
$ mysql -u username -p < bkperfdashboard.sql
```

6) Lancer le server

```
$ nodejs server.js
```

7) Ouvrir l'application dans un navigateur à l'adresse <http://localhost:1337/>

Compte **administrateur** pour créer des comptes client et des utilisateurs :

```
login: perfdashboard@backelite.com  
mdp: *backelite1234#
```

Un utilisateur « client » associé au compte client « Backelite » :

```
login: client@backelite.com  
mdp: 1234
```

Un utilisateur « basique » associé au compte client « Backelite » :

```
login: user@backelite.com  
mdp: 1234
```