API Build and Deployment

Projet : DashyAPINode
By: ADEGNON David
Version : Final
Date : 2017-05-26

Table of Contents

Introduction

Description

Ce document décrit le processus pour le déploiement de l'application 'Dashy' . Dashy est une API écrit en Node.JS permettant d'intégrer les métriques de ChartMogul et Google Analytics sur GeckoBoard et définit aussi plusieurs 'pushers' pour envoyer des datas à intervalles réguliers et définies à des widgets GeckoBoard.

Toutes les informations nécessaires pour configurer et déployer l'API seront détaillées :

- Les étapes de configuration
- Les fichiers ou dépendances à rajouter.
- Comment installer et exécuter l'API.

Et aussi, d'autres étapes informations concernant le répertoire de code et le déploiement.

Cible

Ce document est principalement destiné à une personne possédant des compétences en programmation. Il doit être familier avec l'environnement et le terminal de Linux, posséder un minimum de connaissance en JS.

Organisation du document

- Répertoire du code : Détailles les différents dossiers/fichiers présents dans le dossier du code source
- Configuration : Les différentes étapes pour la configuration de l'API
- Exécution : Comment effectuer le lancement de l'application.
- Processus de déploiement : Comment effectuer le déploiement.
- Architecture : Cette partie n'est pertinent que pour les développeurs et ceux qui souhaitent continuer le développement de l'application.
- Améliorations : Les différents bugs restants, piste pour amélioration, …

Répertoire du code

BitBucket

Le projet est hébergé sur le BitBucket d'Apptweak. Pour pouvoir continuer à develloper ou effectuer un commit, vous aurez besoin d'un client GIT.
Cependant, BitBucket met à disposition un lien https pour obtenir la version ZIP du projet.

- HTTPS

https://DavidStage@bitbucket.org/apptweak/dashy-api-node.git

- GIT

git@bitbucket.org:apptweak/dashy-api-node.git

Détails

- ***.vscode*** : Dossier pour les configs utilisé par l'éditeur Miscrosoft Visual Studio Code
- ***assets*** : Dossier pour les fichiers pour la route /static de l'API
- ***config*** : Dosshiers pour les fichiers de configuration requis pour l'application
- ***scripts*** : Dossier pour les scripts pour le setup
- ***src*** : Dsossier pour les fichiers sources de l'application
- ***test*** : Dossier pour les fichiers de test du module '*components*'.
- ***.editorconfig*** : Fichier de configuration pour le package 'editor' de VS Code.
- ***.eslintrc***.json : Fichier de configuration pour le package 'eslint' de VS Code.
- ***.gitignore*** : Fichier de configuration pour les différents dossiers/fichiers à ignorer lors du commit.
- ***.jsdocrc.json*** : Contient la configuration pour le module node, jsDoc
- ***.pm2.json*** : Contient la configuariont pour le module node, pm2
- ***LICENSE*** : Contient les termes et conditions à respecter pour utiliser et distribuer l'application.
- ***Makefile*** :
- ***README.md*** : un simple fichier README étant une version simplifiée de ce document.
- ***apidoc.json*** : Fichier de configuration pour le module node, apiDoc
- ***nodemon.json*** : Fichier de configuration pour le module node, nodemon.
- ***package.json*** : Fichier contenant tous les informations sur le projet pour npm.

Configuration

Le processus qui s'entame à partir de cette partie se déroule dans un environnement Linux. Certaines commandes ou leurs effets pourraient différer si l'environnement de votre système d'exploitation n'est également Linux.

Avant de commencer, récupérer le code source.

Pour la configuration de l'application, 2 possibilités sont envisageables :

- Créer, à la racine du dossier, un fichier nommé '***.env***'.
- Créer, dans le dossier '***config'***, un fichier nommé '***production.json***'.

En aucun cas, les 2 fichiers doivent être crées, seulement un des 2.

Les 2 possibilités seront détaillées.

Le fichier '***production.json***' est un fichier JSON et doit commencer par '{ }'. Doit contenir obligatoirement contenir '***api'***, '***redis'***, '***geckoBoard'***, '***chartMogul'***, '***google-analytics'*** et sont tous des objets.

```
{
    "api": {},
    "redis": {},
    "geckoBoard": {},
    "chartMogul": {},
    "google-analytics": {}
}
```

*Figure 1 - Aperçu de **production.json** à la création*

Voici le template pour le fichier '**production.json**' avec la transposition des valeurs pour le fichier des variables d'environnements du fichier '***.env***'

```
{
    "api": {
        "port": "API_PORT",
        "token": "API_TOKEN",
        "dirLogs": "API_DIR_LOGS"
    },
    "redis": {
        "host": "REDIS_HOST",
        "port": "REDIS_PORT",
        "password": "REDIS_PWD",
        "db": "REDIS_DB"
    },
    "geckoBoard": {
        "apiKey": "GECKO_API_KEY",
        "pushTime": "WIDGETS_DEFAULT_PUSH_TIME",
        "widgets": {
            "leadsMonth": {
                "id": "WIDGET_LEADS_MONTH",
                "pushTime": "WIDGET_LEADS_MONTH_TIME"
            },
            "leadsToday": {
                "id": "WIDGET_LEADS_TODAY",
                "pushTime": "WIDGET_LEADS_TODAY_TIME"
            },
            "latestCustomers": {
                "id": "WIDGET_LAST_CUSTOMERS",
                "pushTime": "WIDGET_LAST_CUSTOMERS_TIME"
            },
            "latestLeads": {
                "id": "WIDGET_LAST_LEADS",
                "pushTime": "WIDGET_LAST_LEADS_TIME"
            },
            "currentActiveUsers": {
                "id": "WIDGET_CURRENT_ACTIVE_USERS",
                "pushTime": "WIDGET_CURRENT_ACTIVE_USERS_TIME"
            }
        }
    },
    "chartMogul": {
        "apiToken": "CHARTMOGUL_API_TOKEN",
        "apiSecret": "CHARTMOGUL_API_SECRET"
    },
    "google-analytics": {
        "viewId": "GA_VIEW_ID",
        "auth": {
            "iss": "GA_SERVICE_EMAIL",
            "keyFile": "GA_KEY_FILE_PATH"
        }
    }
}
```

Figure 2 - Template pour production.json avec correspondance pour .env

Figure 3 - Valeurs par-défaut

```
{
    "api": {
        "port": 8080,
        "token": "API_TOKEN",
        "dirLogs": "./logs"
    },
    "redis": {
        "host": "127.0.0.1",
        "port": 6379,
        "password": "pwd",
        "db": 5
    },
    "geckoBoard": {
        "apiKey": "GECKO_API_KEY",
        "pushTime": 60,
        "widgets": {
            "leadsMonth": {
                "id": ""
            },
            "leadsToday": {
                "id": ""
            },
            "latestCustomers": {
                "id": ""
            },
            "latestLeads": {
                "id": ""
            },
            "currentActiveUsers": {
                "id": ""
            }
        }
    },
    "chartMogul": {
        "apiToken": "CHARTMOGUL_API_TOKEN",
        "apiSecret": "CHARTMOGUL_API_SECRET"
    },
    "google-analytics": {
        "viewId": 1234567890,
        "auth": {
            "iss": "GA_SERVICE_EMAIL",
            "keyFile": "./google-credentials.p12"
        }
    }
}
```

Api

'api' regroupe toutes les valeurs pour configurer l'application. Voici les détails :

production.json	.env	Par Défaut	Description
port	API_PORT	`8080`	Le port sur lequel l'application écoute
host	API_HOST		
token	API_TOKEN	`"TOKEN_FOR_API"`	Le token à rajouter à l'URL pour autoriser l'exécution de la requête
dirLogs	API_DIR_LOGS	`"./logs"`	Le dossier pour enregistrer les fichiers '***.log***'

Redis

'redis' regroupe toutes les valeurs pour la connexion vers votre DB REDIS. Voici les détails :

production.json	.env	Par Défaut	Description
host	REDIS_HOST	`"127.0.0.1"`	L'adresse IP/URL du serveur REDIS
port	REDIS _PORT	`63079`	Le port du serveur REDIS
pwd	REDIS _PWD	`"pwd"`	Le token à rajouter à l'URL pour autoriser l'exécution de la requête
db	REDIS _DB	`0`	Le numéro de database à utiliser sur le serveur REDIS

A ce stade, votre fichier '***production.json***' ou '***.env***' devrait ressembler à ceci mais avec les valeurs par défaut :

```
{
    "api": {
        "port": 8080,
        "token": "API_TOKEN",
        "dirLogs": "./logs"
    },
    "redis": {
        "host": "127.0.0.1",
        "port": 6379,
        "password": "pwd",
        "db": 5
    },
    "geckoBoard": {},
    "chartMogul": {},
    "google-analytics": {}
}
```

Figure 4 - Aperçue de 'production.json'

```
API_PORT=8080
API_TOKEN=API_TOKEN
API_DIR_LOGS=./logs
REDIS_HOST=127.0.0.1
REDIS_PORT=6379
REDIS_PWD=pwd
REDIS_DB=0
```

Figure 5 - Aperçue du '.env'

GeckoBoard

'geckoBoard' regroupe toutes les valeurs propres à la partie GeckoBoard de l'API.

production.json	.env	Par Défaut	Description
apiKey	GECKO_API_KEY	`"GECKO_API_KEY"`	La clé fournie par GeckoBoard pour l'envoi des données.
pushTime	WIDGETS_DEFAULT_PUSH_TIME	`60`	Le nombre de secondes entre chaque pushing pour tous les widgets en mode '***push***'.

'geckoBoard' contient une sous-partie pour accueillir les configurations pour les widgets en mode 'push'. Chaque widget a son objet JSON contenant l'***id*** et le ***pushTime***.
Par défaut, le Dashboard comprend 5 widgets en mode '***push' :***

- ***Les leads du mois***
- ***Les leads apparus aujourd'hui***
- ***Les derniers leads***
- ***Les derniers Customers***
- ***Le nombre de visiteurs actuellement sur apptweak.com***

Pour chacun de ces widgets, aucune valeur par-défaut n'est défini donc leur id doit **obligatoirement** être dans votre fichier de configuration et éventuellement spécifié un '***pushTime'***.

Exemple de configurations utilisées durant le développement :

```
"geckoBoard": {
    "apiKey": "GECKO_API_KEY",
    "pushTime": 60,
    "widgets": {
        "leadsMonth": {
            "id": "144091-6b060040-f61f-0134-9c3b-22000b4a867a",
            "pushTime": 45
        },
        "leadsToday": {
            "id": "144091-122cb660-f785-0134-c10c-22000b248df5",
            "pushTime": 45
        },
        "latestCustomers": {
            "id": "144091-4946dac0-fdc8-0134-566c-22000b498417",
            "pushTime": 90
        },
        "latestLeads": {
            "id": "144091-117deab0-fdbe-0134-6def-22000abade87",
            "pushTime": 90
        },
        "currentActiveUsers": {
            "id": "144091-4dfd23d0-1c4e-0135-5be0-22000a891cb1",
            "pushTime": 3
        }
    }
},
```

*Figure 6 - Partie 'geckoBoard' dans **production.js***

```
GECKO_API_KEY=KEY_FOR_GECKO
WIDGETS_DEFAULT_PUSH_TIME=90
WIDGETS_LEADS_MONTH=144091-6b060040-f61f-0134-9c3b-22000b4a867a
WIDGETS_LEADS_MONTH_TIME=45
WIDGETS_LEADS_TODAY=144091-122cb660-f785-0134-c10c-22000b248df5
WIDGETS_LEADS_TODAY_TIME=45
WIDGETS_LAST_CUSTOMERS=144091-122cb660-f785-0134-c10c-22000b248df5
WIDGETS_LAST_CUSTOMERS_TIME=90
WIDGETS_LAST_LEADS=144091-117deab0-fdbe-0134-6def-22000abade87
WIDGETS_LAST_LEADS_TIME=90
WIDGETS_CURRENT_ACTIVE_USERS=144091-4dfd23d0-1c4e-0135-5be0-22000a891cb1
WIDGETS_CURRENT_ACTIVE_USERS_TIME=3
```

*Figure 7 - Equivalent dans **.env***

ChartMogul

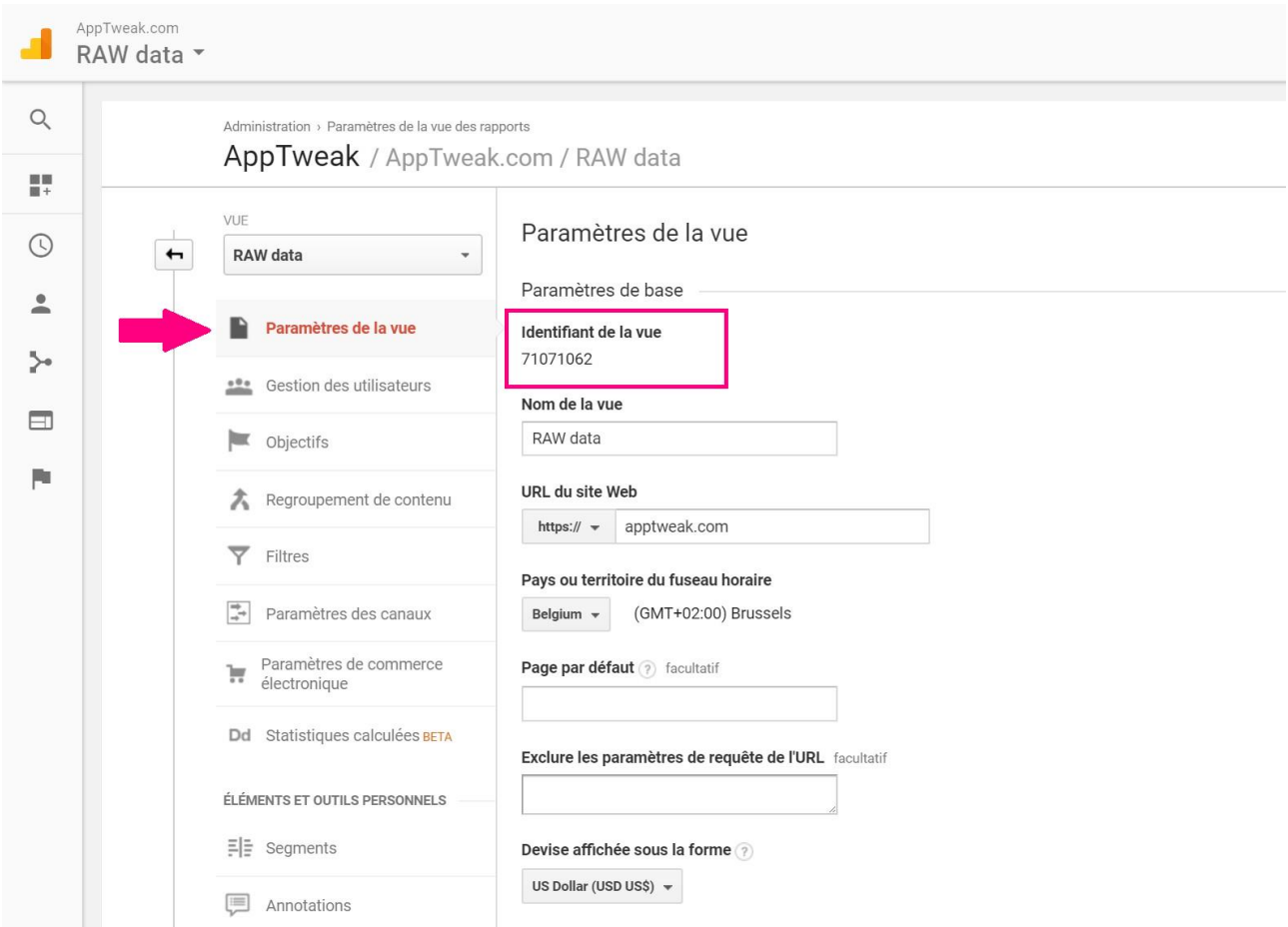
'chartMogul' regroupe toutes les valeurs propres à la partie ChartMogul de l'API.

production.json	.env	Par Défaut	Description
apiToken	CHARTMOGUL_API_TOKEN	`"TOKEN_FOR_CHARTMOGUL"`	Le token d'accès pour l'API de ChartMogul
apiSecret	CHARTMOGUL_API_SECRET	`"SECRET_FOR_CHARTMOGUL"`	Le secret pour l'API de ChartMogul.

Google Analytics

production.json	.env	Par Défaut	Description
viewId	GA_VIEW_ID	`1234567890`	L'id de la vue/profil dans Google Analytics.

Cet "viewID", nommé « Identifiant de la vue », est disponible sur la page de 'Administration' de Google Analytics. Dans ce cas, la vue en question est « Apptweak.com/RAW DATA ».



Google APIS pour accéder à l'API de Google Analytics requiert un compte de service.
Ce compte de service peut être créer dans la [console développeurs de Google](#).

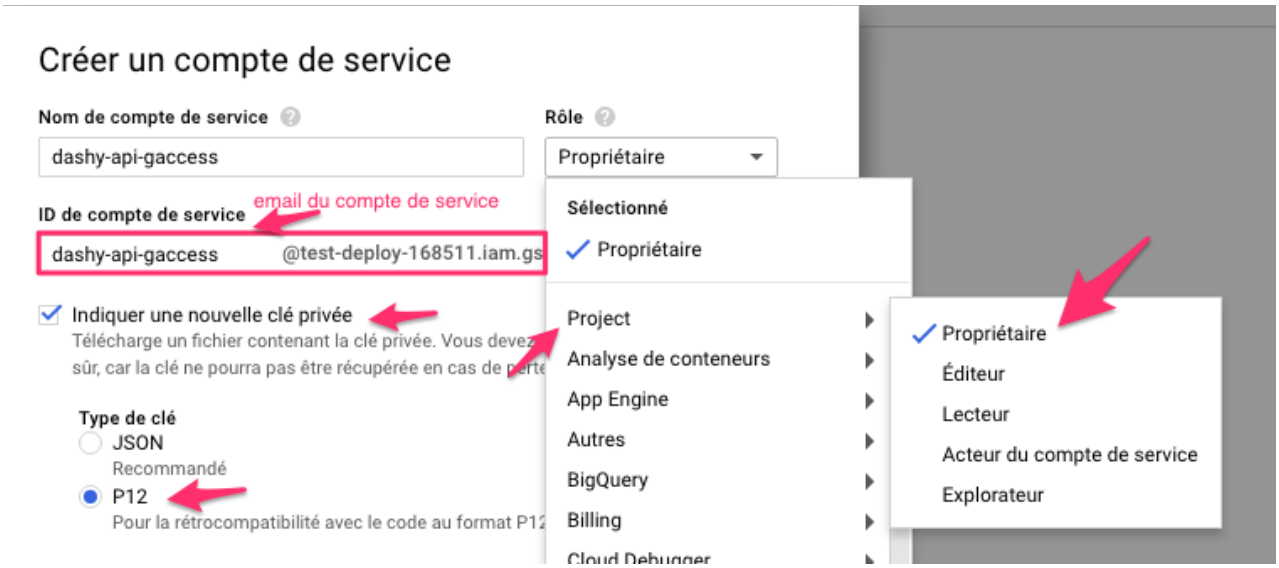
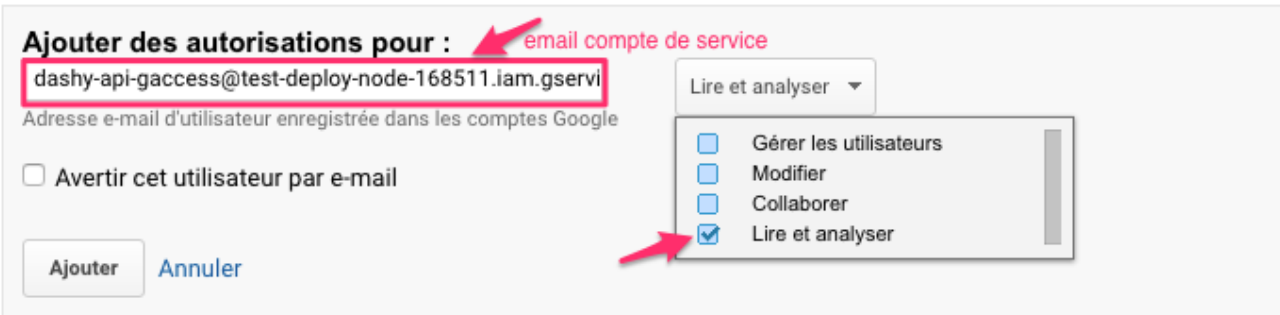


Figure 8 - Aperçu des actions pour la création du compte de servie

Lors de la création du compte de service, un fichier '***.p12***' va être téléchargé. Elle contient la clé privée pour authentifier les requêtes vers Google APIS.
Ce fichier peut aussi être du type '.json' voire '.pem'.

Durant le développement, j'ai placé ce fichier dans le dossier '***config***' et renommée en '***google-credentials.p12'.***
L'email fournie doit être, aussi rajouter, parmi les utilisateurs de Google Analytics et avoir au moins les permissions de 'Lire et analyser'.



Ses deux valeurs de configuration (***'iss'*** et '***keyFile'***) se trouvent dans la sous-partie '***auth'***.

production.json	.env	Par Défaut	Description
iss	GA_SERVICE_EMAIL	`"GA_SERVICE_EMAIL"`	L'email de service
keyFile	GA_KEY_FILE_PATH	`"./google-credentials.p12"`	Où trouver le fichier de credentials par rapport au dossier config.

Exemple de configurations utilisées durant le développement :

```
"chartMogul": {
    "apiToken": "TOKEN_CHARTMOGUL_API",
    "apiSecret": "SECRET_CHARTMOGUL_API"
},
"google-analytics": {
    "viewId": 71071062,
    "auth": {
        "iss": "dashyaccessga@dashy.iam.gserviceaccount.com",
        "keyFile": "./config/google-credentials.p12"
    }
}
```

*Figure 9 - Parties 'chartMogul' et 'google-analytics' dans **production.json***

```
CHARTMOGUL_API_TOKEN=TOKEN_FOR_CHARTMOGUL
CHARTMOGUL_API_SECRET=SECRET_FOR_CHARTMOGUL
GA_VIEW_ID=71071062
GA_SERVICE_EMAIL=dashyaccessga@dashy.iam.gserviceaccount.com
GA_KEY_FILE_PATH=./config/google-credentials.p12
```

*Figure 10 - Equivalent pour **.env***

Exécution

Vous voilà, après une configuration effectuée, prêts pour la suite.
Il ne reste plus qu'à lancer l'application.

Préparation

Vous devez vous trouver dans un environnement Linux.
Votre version de Node.JS doit être >= **v7.8.0**
Dans un terminal et à la racine du projet, exécuter la commande suivante :

```
npm  install && npm start
```

La première commande installe tous les modules nécessaires au fonctionnement de l'application ensuite lance l'application.

Durant l'exécution du '***npm start***', 2 actions sont effectuées :

1. Il définit les variables d'environnement :
    a. ***TZ=Europe/Brussels***
    Définit le time zone de l'application. Par défaut, aucune time zone n'est définie et l'heure fournie par Javascript est en UTC (Universal Time Zone).
    b. ***NODE_ENV=production***
    Une des conventions en Node.JS est la modification du contexte de l'application selon la valeur de la variable '***NODE_ENV***'.
2. Le module PM2 effectue l'exécution du fichier '***app.js***' se trouvant dans le dossier '***src***'.

Processus de déploiement

Architecture

Ajout d'une nouvelle Data Source

Pour une meilleure compréhension, on va détailler le processus de gsetion pour une Data Source (Wootric).
La Data Source est Wootric et la data souhaitée est le NPS.

Configurations pour Wootric

a. Rajouter une partie 'wootric' dans le fichier 'default.json' du dossier 'config'. Rajouter l'URL pour contacter l' API de Wootric.
b. Puis, rajouter cette même partie dans le fichier 'custom-environment-variables.json' pour créer des variables d'environnement.
c. Ensuite, rajouter cette même partie dans le fichier 'production.json' mais ne conserver que la partie qui risque de changer tel que le client ID et secret.
d. Faire de même pour le fichier 'test.json' et en changeant aussi l'URL par une fausse adresse afin d'éviter.

```
"wootric" : {
  "authUrl": "https://api.wootric.com/oauth/token",
  "apiUrl": "https://api.wootric.com/v1",
  "grantType" : "client_credentials",
  "clientId": "TOKEN_FOR_WOOTRIC_API",
  "clientSecret": "SECRET_FOR_WOOTRIC_API"
},
```

Figure 12 - Point 1.a

```
"wootric": {
  "clientId": "WOOTRIC_API_ID",
  "clientSecret": "WOOTRIC_API_SECRET"
}
```

Figure 11 - Point 1.b

```
"wootric" : {
  "authUrl": "https://api.wootric.mock/oauth/token",
  "apiUrl": "https://api.wootric.mock/v1",
  "clientID": "WOOTRIC_CLIENT_ID",
  "clientSecret": "WOOTRIC_CLIENT_SECRET",
},
```

Figure 13 - Point 1.d

Feeder

e. Créer une classe WootricFeeder héritant de la classe Feeder se trouvant dans le dossier '**components'.**
f. Dans le constructor de cette nouvelle classe, appeler le constructeur du parent avec paramètre l'URL defini dans le fichier de Configuration via le module '**config'**.
g. Vous pouvez réécrire la fonction '**requestAPI'** avec les mêmes arguments que dans la fonction originelle.
h. Ou dans mon cas, créer une fonction '**requestWootricFor'** s'occupant de contacter l'API pour recevoir les data.
i. Aussi, créer une fonction '**requestWootricForToken'** pour recevoir un token d'accès.
j. Pour chaque route disponible dans cette partie Wootric, créer une fonction Fetcher. Chaque fetcher, doit avoir son nom qui commence par 'fetch' suivi par le nom de la data à recevoir.
   Example : Pour la data du WootricFeeder, la fonction doit se nommer 'fetchNPS'.
k. Tous les Fetchers, reçoivent de la part du router en arguements, un objet '**config'** contenant les paramètres pour le Fetch.

Router

a. Créer une classe WootricRouter héritant de la classe BaseRouteur se trouvant dans le dossier '**base'.**
b. Dans le constructor de cette nouvelle classe, appeler le constructeur du parent avec paramètre, le WootricFeeder suivi par l'URL defini pour cette route.

```
class WootricRouter extends BaseRouter {
  /**
   * Creates an instance of WootricRouter.
   * @param {module:feeder/Wootric} feed - The feeder for this router.
   *
   */
  constructor(feed) {
    super(feed, '/wootric');
  }


  /** @override */
  handler() {
    super.handler();
```

Figure 14 - Début du WootricRouter

c. 2 méthodes doivent être réécrite : '**handler'** et '**handlerPusher'**.
d. La fonction '**handler'** contient tous les routes et comment répondre à la requête.
e. Cette fonction doit obligatoirement appeleer la méthode handler du parent.

```
/** @override */
handler() {
  super.handler();  3.e


  /*⟺*/


  // LeaderBoard of 5 Most Plans Subscribed
  /*⟺*/

  this.router_.get('/nps', async(ctx, next) => {
    ctx.state.data = await this.feed_.fetchNPS(ctx.state.config);
    return next();
  });
}
```

f. Les routes s'écrivent de cette manière. Elle reçoivent la sous-route à gérer et le middleware à appeler pour y répondre.
g. Le middleware est toujours de type de async et a 2 arguements : ctx et next :
    i. Pour passer les données à retourner, celle-ci doivent être assigner dans la variable 'ctx.state'.
    ii. 'await' doit être utilisé car la fonction 'fetchNPS' est de type async.
    iii. Et fin du middleware, on doit toujours retourner le next en arguments.

h. La fonction **'handlerPusher'** est pour les widgets en mode push.
i. Cette partie Wootric du Dashboard ne contient aucun widgets en mode Push. Je vais illustrer la démarche à suivre en me servant de ChartMogul.

```
/** @override */
handlerPusher() {
  const widgets = Config.geckoBoard.widgets;

  [
    [
      widgets.leadsMonth.id, // Leads Month
      this.feed_.fetchNbLeads,
      widgets.leadsMonth.pushTime,
    ],
    [⟺],
    [⟺],
    [
      widgets.latestLeads.id, // Last Leads Month
      [this.feed_.fetchLatestCustomers, {onlyLead: true}],
      widgets.latestLeads.pushTime,
    ],
  ].forEach((p) => this.listPushers_.push(new Pusher(...p, this.feed_)));
};
```

Figure 15 - Example de handlePusher pour ChartMogul

j. Chaque Routeur contient une variable protected 'this.listPushers_'.
   Cette variable est un tableau de Pusher qui doit être remplie de Pusher.

k. Chaque pusher doit être créer avec en paramètres :

   iv. l'ID du widget en mode Push.
   Dans mon cas, je regroupe tous les widgets en mode par leur nom dans le fichier de configuration **'production.json'**.

```
"geckoBoard": {
  "apiKey": "d9ec39b3c32aa824655fe7f3a2f197b9",
  "pushTime": 90,
   "widgets" : {
     "leadsMonth" : {⟺},
     "leadsToday": {
       "id" : "144091-122cb660-f785-0134-c10c-22000b248df5",
       "pushTime": 40
     },
     "latestCustomers" : {⟺},
     "latestLeads" : {
       "id" : "144091-117deab0-fdbe-0134-6def-22000abade87",
       "pushTime": 90
     },
     "currentActiveUsers": {
       "id": "144091-4dfd23d0-1c4e-0135-5be0-22000a891cb1",
       "pushTime": 5
     }

   }
},
```

Figure 16 - Config de widgets dans production.json

   v. La fonction à appeler pour obtenir les données
   vi. Un Temps de push prédéfini pour ce widget ou alors, **null**.
   vii. La classe de feed qui servira de contexte lors de l'appel de la fonction.

Boot

a. Vient maintenant la dernière partie, concernant le Boot.
b. Il suffit d'initialiser le WootricRouter en lui passant en arguement le WootricFeeder.
c. Passer cette initialisation dans la variable routes_

```
const routes_ = [
  BaseRouter.getInstance(),
  new ChartMogulRouter(new ChartMogulFeeder()),
  new GARouter(new GAFeeder()),
  new WootricRouter(new WootricFeeder()),
];
```

Sécurité

La sécurité n'est pas très élaborée. L'API étant privée et n'ayant une visibilité limitée à ses principaux acteurs à savoir le développeur, l'équipe d'Apptweak.
Pour authentifier une requête vers l'application, il suffit d'ajouter dans les paramètres de la requête, ' ***?token***' avec la valeur définie comme token dans les configurations de l'application. Pour l'heure, cette option est désactivée. Aucun token ne doit être fournie.

Pour l'activer, il faut décommenter dans le fichier BaseRouter.js du dossier base, la ligne **162** et **166**.

Améliorations

1. Changer la manière d'initialiser les composants.

   Chaque nouvelle Data Source doit posséder une router et feeder.

   Ceux-ci sont initialisés au démarrage de l'application dans le fichier '***Boot***'.

   Pour le moment, chaque feeder et routeur sont hard codés dans '***Boot'***

   L'idéal serait d'utiliser un pattern de création pour ces feeders et routeurs.
2. Renforcer la sécurité en rajoutant une page dans l'application dédiée.

Cette page permettrait, après connexion et authentification d'un admin, de créer/changer un mot de passe et recevoir en retour un token.

3. La gestion des fichiers '***.log***' laisse à désirer et doit être amélioré.

   Établir un système de rotation de ces fichiers.