focus.

focus.

AA2 - Bilan d'architecture

focus@epitechfr.onmicrosoft.com
http://eip.epitech.eu/2019/focus

Résumé

Ce document contient une description aussi exhaustive que possible du plan technique de l'EIP Focus, solution de gestion de productivité. En outre, il présente de façon hiérarchisée les différentes briques technologiques du produit.

Focus se décompose en quatre grandes parties : le Backend, le Frontend web, le Frontend mobile et enfin le Daemon. Ces quatre services agissent de concert afin de créer un service de gestion de productivité innovant. Le Daemon est la principale source de données qui propulse Focus. C'est un service transparent installé sur les ordinateurs de l'utilisateur final. Il collecte des données sur l'utilisation des appareils et transmet en continu des métriques clés au Backend de Focus. De plus, le Daemon est en mesure de proposer une gestion poussée des nuisances numériques à l'aide de filtres définis par l'utilisateur. Celui-ci va analyser en profondeur toutes les informations qui lui sont transmises. Il met en relation une masse importante de données disponibles par agrégation afin d'aboutir à une analyse poussée et sur mesure, en cohérence avec les usages de l'utilisateur. Les Frontend web et mobile mettent à disposition des utilisateurs les données récoltées sur un navigateur web et sur leurs smartphones respectivement.

Description du document

Propriétés	Métadonnées
Titre	AA2
Date	03/12/2017
Auteurs	Enzo AGUADO, Hippolyte BARRAUD Robin CHARPENTIER, Dimitri MAS, Come MURE-RAVAUD, Etienne PASTEUR
Version du modèle	1.0
Mots clés	Focus, EIP, Epitech, AA, AA2, Bilan, Architecture

Table de révisions

Date	Version	Section(s)	Auteurs	Commentaires
20/02/2017	v0.1	Toutes	Etienne PASTEUR	Création initiale
21/11/2017	v1.0	Toutes	Enzo AGUADO, Hippolyte BARRAUD Robin CHARPENTIER, Dimitri MAS, Come MURE-RAVAUD, Etienne PASTEUR	Création du contenu

Table des matières

Rappel de l'EIP

Qu'est-ce qu'un EIP et Epitech

Epitech (École pour l'informatique et les nouvelles technologies) est un établissement d'enseignement privé français créé en 1999 par le groupe IONIS qui délivre un enseignement supérieur en informatique et nouvelles technologies.

L'école propose à ces étudiants, à partir de leur troisième année, un projet de fin d'études : l'EIP (pour Epitech Innovative Project).

A ce titre, les élèves doivent s'organiser en un groupe d'au moins six personnes et choisir un sujet innovant. L'EIP est un passage obligatoire et unique dans la scolarité de l'étudiant, de par son envergure (18 mois) et la préparation requise.

Un Epitech Innovative Project est conçu (comme toute la pédagogie Epitech fondée sur la méthode projets) à la manière d'un véritable projet entrepreneurial, dans toutes ses composantes : business, techno, design & communication. Un EIP est appelé à devenir une start-up viable à la fin du cursus de l'étudiant.

Sujet de l'EIP Focus

Focus est une solution visant à améliorer le workflow de travail des utilisateurs en leur donnant plus de temps pour faire ce qui compte vraiment.

Grace a des rapports d'activité et des analyses avancées fournis par des outils installés sur vos ordinateurs et smartphones, Focus propulse des informations pertinentes sur votre activité. Un *reporting* complet et détaillé de vos activités quotidiennes vous offre un aperçu global de l'organisation de votre journée, vous permettant de mieux identifier les tâches les plus chronophages.

Ces données permettent à Focus de comprendre comment vous travaillez et de regrouper les sessions de travail analogues (répondre à des courriels, rencontrer des clients, téléphoner, etc.), capturer les notifications en fonction de filtres intelligents et de rationaliser globalement le flux de travail.

Le but ultime est de permettre à nos utilisateurs d'être 100% focalisés sur une tâche unique et bien définie à la fois et de les préserver de toute distraction pendant qu'ils réalisent cette tâche.

Représentation de l'architecture globale

Focus est composé de trois services majeurs : un daemon installé sur l'ordinateur de l'utilisateur relevant son activité, un backend traitant ces informations afin de leur donner du sens, et un frontend offrant une interface sur ces données en plus d'offrir des possibilités de configuration et d'administration.

Ce document traite donc naturellement ces trois éléments d'un point de vue différent. Chaque plateforme a des use cases et besoins différents, et requiert par conséquence une architecture différente.

Aujourd'hui, Focus adopte un pattern de Microservices. Chaque composant est découplé des autres afin d'assurer une interopérabilité et une flexibilité cruciales à la bonne évolution du projet sur le long terme. Ce document détaille l'architecture de ces différents services.

Frontend :

Basé sur des technologies web, le frontend est l'interface entre l'utilisateur et le cœur de Focus. Cette interface propose à l'utilisateur de s'inscrire, s'authentifier, accéder à ses paramètres, gérer son profil, et, essentiellement, de retracer ses activités passées. Le second atout du frontend est de permettre la configuration des filtres à nuisances.

Ces derniers forment la deuxième fonctionnalité clé de Focus, le contrôle des nuisances numériques (notifications, emails, messages, etc), donnant à l'utilisateur la possibilité d'instaurer des règles pour bloquer, prioriser ou retarder les informations rentrantes pendant un certain temps.

Le frontend communique avec le serveur web de manière bidirectionnelle.

Daemon :

Le Daemon est l'œil de Focus. Sous forme d'un logiciel discret à installer sur les ordinateurs de l'utilisateur, le Daemon récupère l'activité de l'utilisateur grâce aux APIs spécifiques à chaque plateforme :

Linux : *X window API*

Mac : *Cocoa, Carbon et Quartz*

Windows : *Win32 API*

Ces informations sont dans un premier temps stockées dans une queue locale en mémoire RAM. Elles passent ensuite par une préanalyse qui va rassembler les informations complémentaire et/ou

redondantes, les compresser, les sérialiser, et enfin les envoyer au backend pour un traitement plus approfondi.

Le transport se fait au travers de sockets fournis par la librairie *NanoMsg*, sécurisés par le protocole TLS et authentifiés par un système de tokens générés à la connexion du Daemon.

Backend :

Le backend joue le rôle du cerveau de Focus. Il a pour charge de recueillir les informations envoyées par le(s) Daemon(s), de les analyser avec plusieurs niveaux de précision, et de les persister dans une base de données NoSQL Apache Cassandra.

A la réception des données du daemon, le backend va créer un flux passant par plusieurs modules d'analyse. Chaque module diffère par la nature de l'information qui en sortira : un module se chargera de savoir sur quel logiciel l'utilisateur passe le plus de temps, un autre déterminera quel projet ou tâche se révèle être chronophage, etc. Cette modularité offre des perspectives d'évolution puissantes et variées.

Ce flux se termine dans une base de données distribuée, garantissant une tolérance aux pannes et une disponibilité importante.

Vue globale du projet

Use-cases principaux

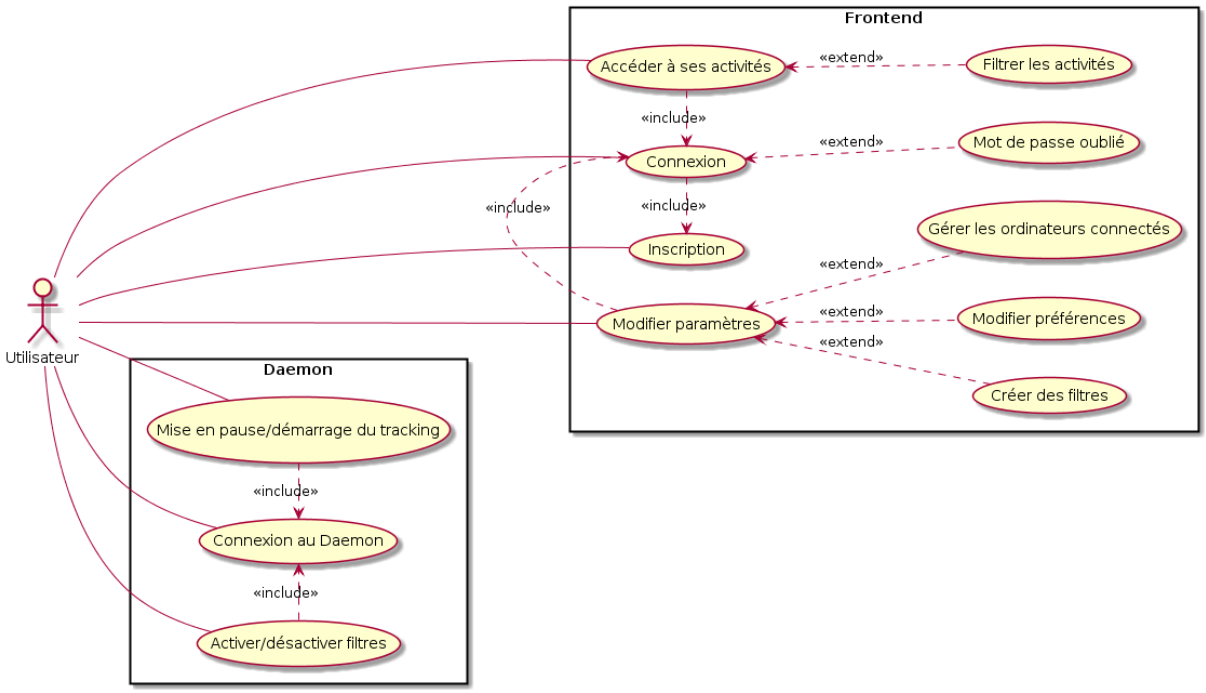

Figure 1 - Diagramme UML use case global

Le frontend est l'interface par laquelle communique principalement l'utilisateur. Le choix d'offrir une plateforme web plutôt que de tout afficher dans une fenêtre gérée par le Daemon présente plusieurs avantages. Les mises à jour sont mieux contrôlées, plus simples et plus rapides. Les coûts et temps de développement et la maintenance sont grandement réduits. L'accessibilité du produit est aussi grandement améliorée.

Use-Cases détaillés

Frontend

Les cas d'utilisations présentés dans cette section concernent le frontend web et mobile. Les deux plateformes présenteront les mêmes fonctionnalités :

- Authentification (inscription, connexion, mot de passe oublié)
- Accéder à son profil (voir son activité)
- Crée et modifier des filtres permettant de gérer les différentes nuisances numériques
- Gérer ses paramètres personnels

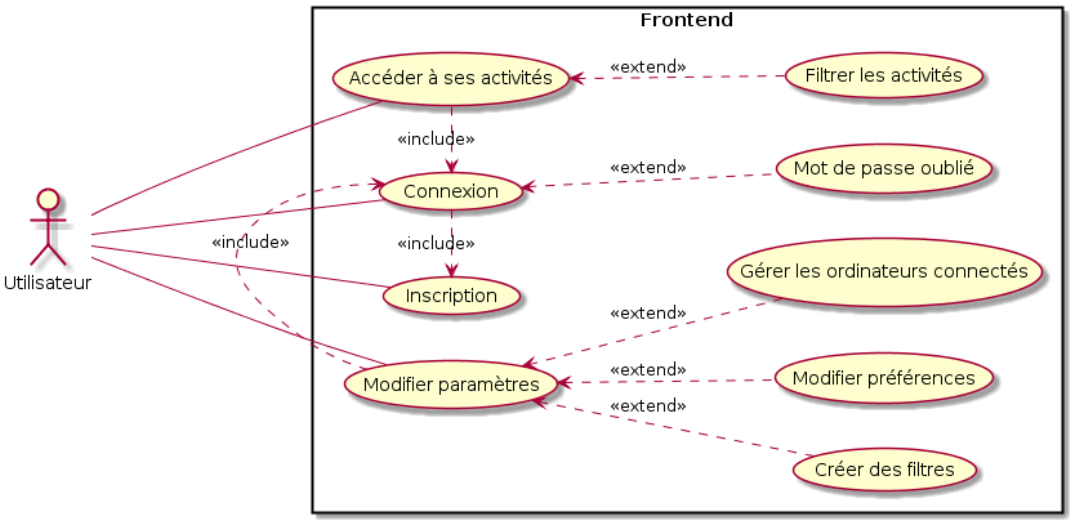

Figure 3 - Use-cases du frontend

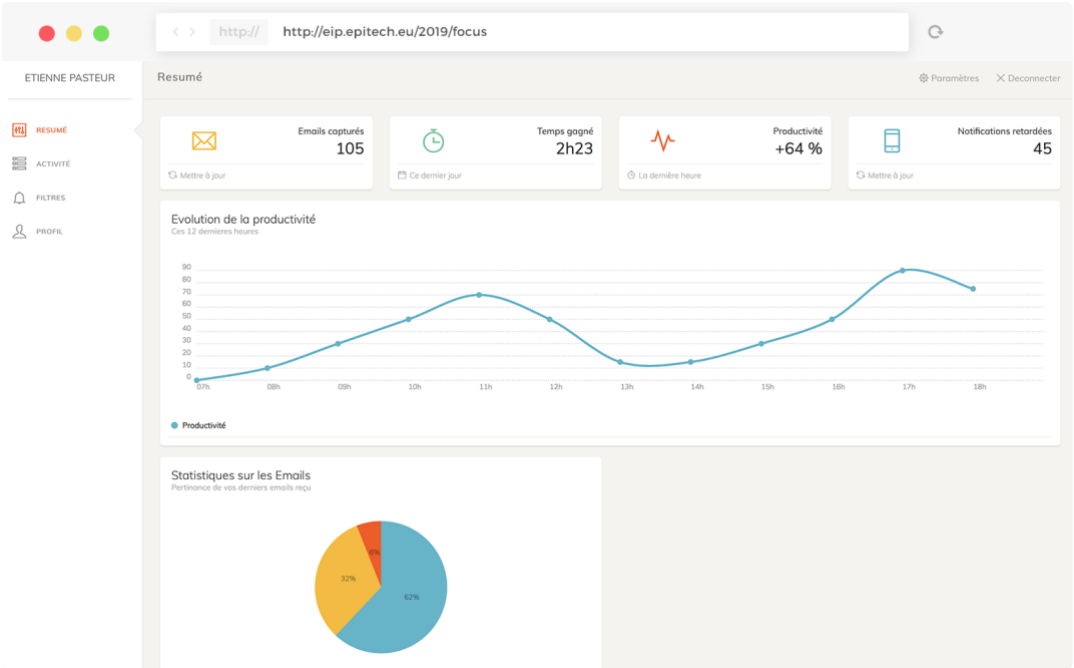

Figure 2 - Maquette du frontend web (vue résumé)

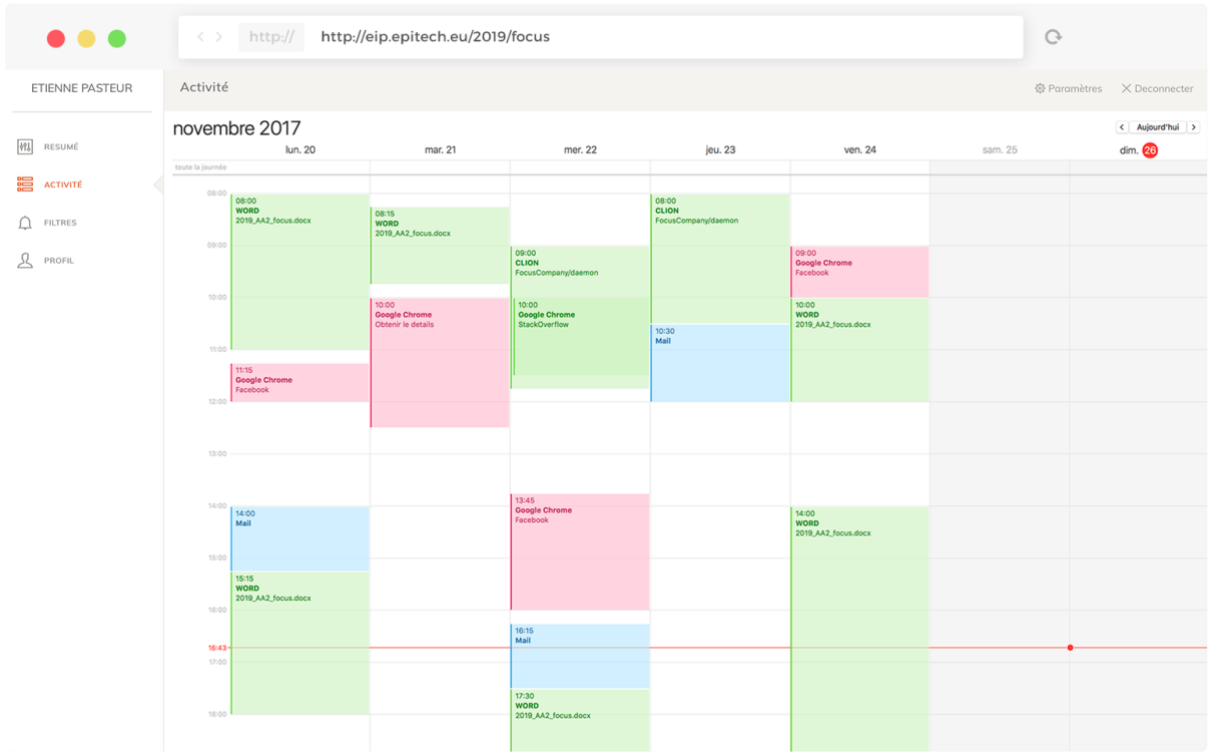

Figure 4 - Maquette du frontend web (vue activité)

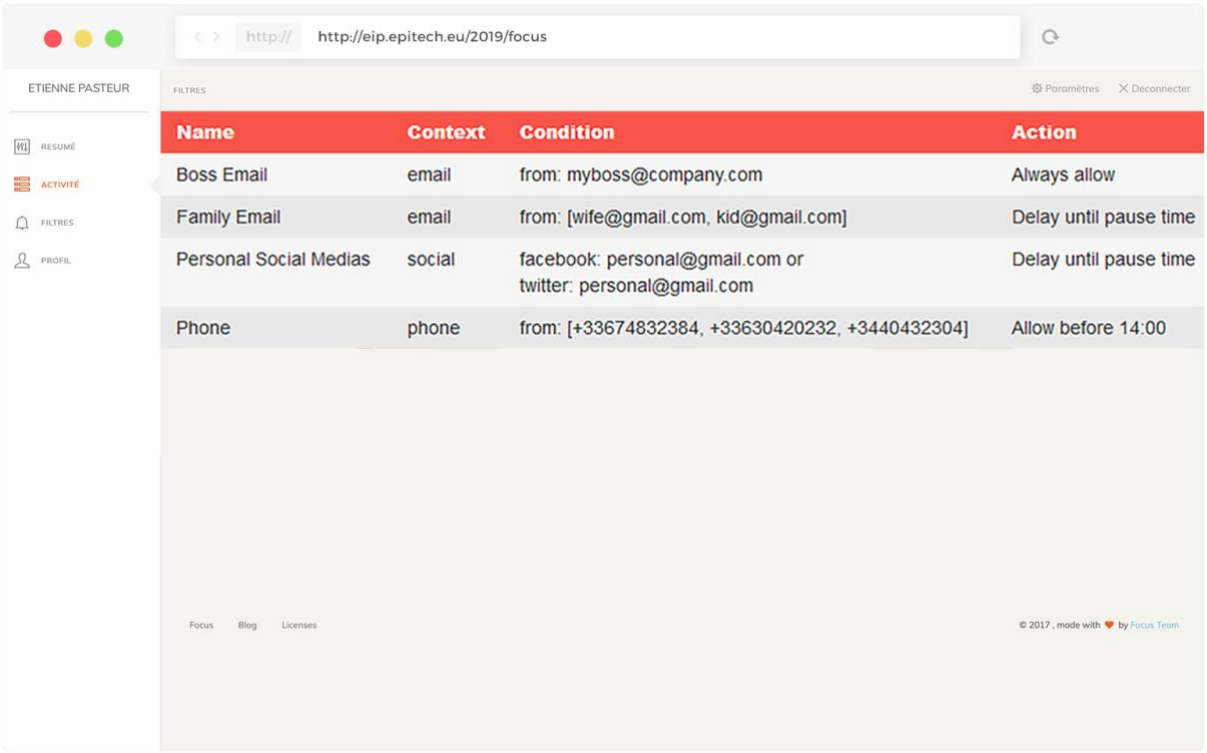

Figure 5 - Maquette du frontend web (vue gestion de filtres)

Daemon

Les cas d'utilisation du daemon sont simples :

- Se connecter au daemon
- Contrôler l'activité du daemon (lancer le monitoring, le stopper)
- Activer/désactiver les filtres à nuisances

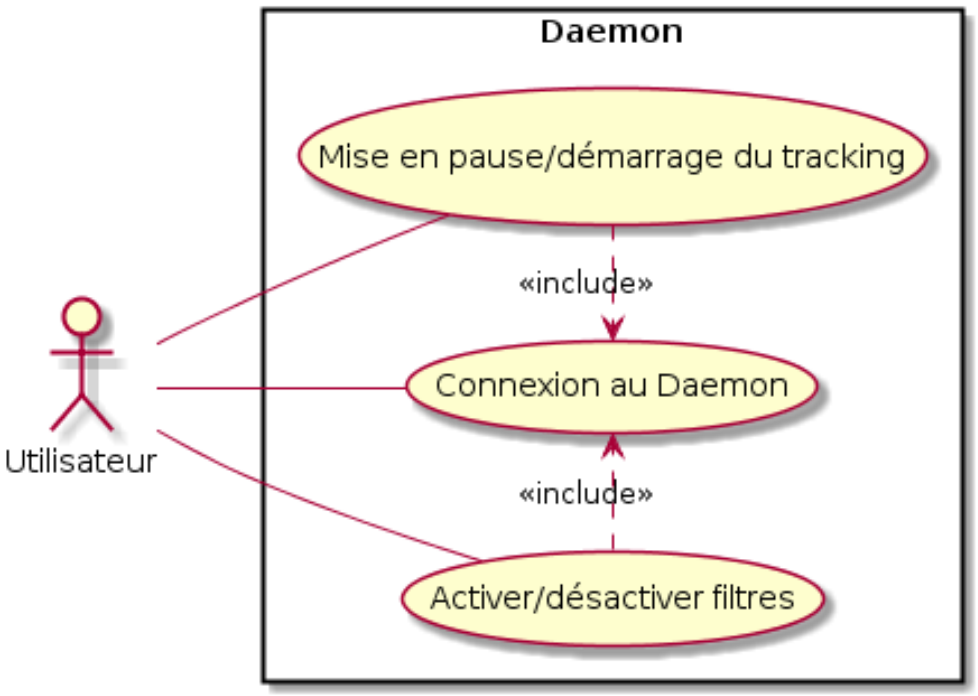

Figure 6 - Use-cases du daemon

Vue Logique de l'application

Vue globale

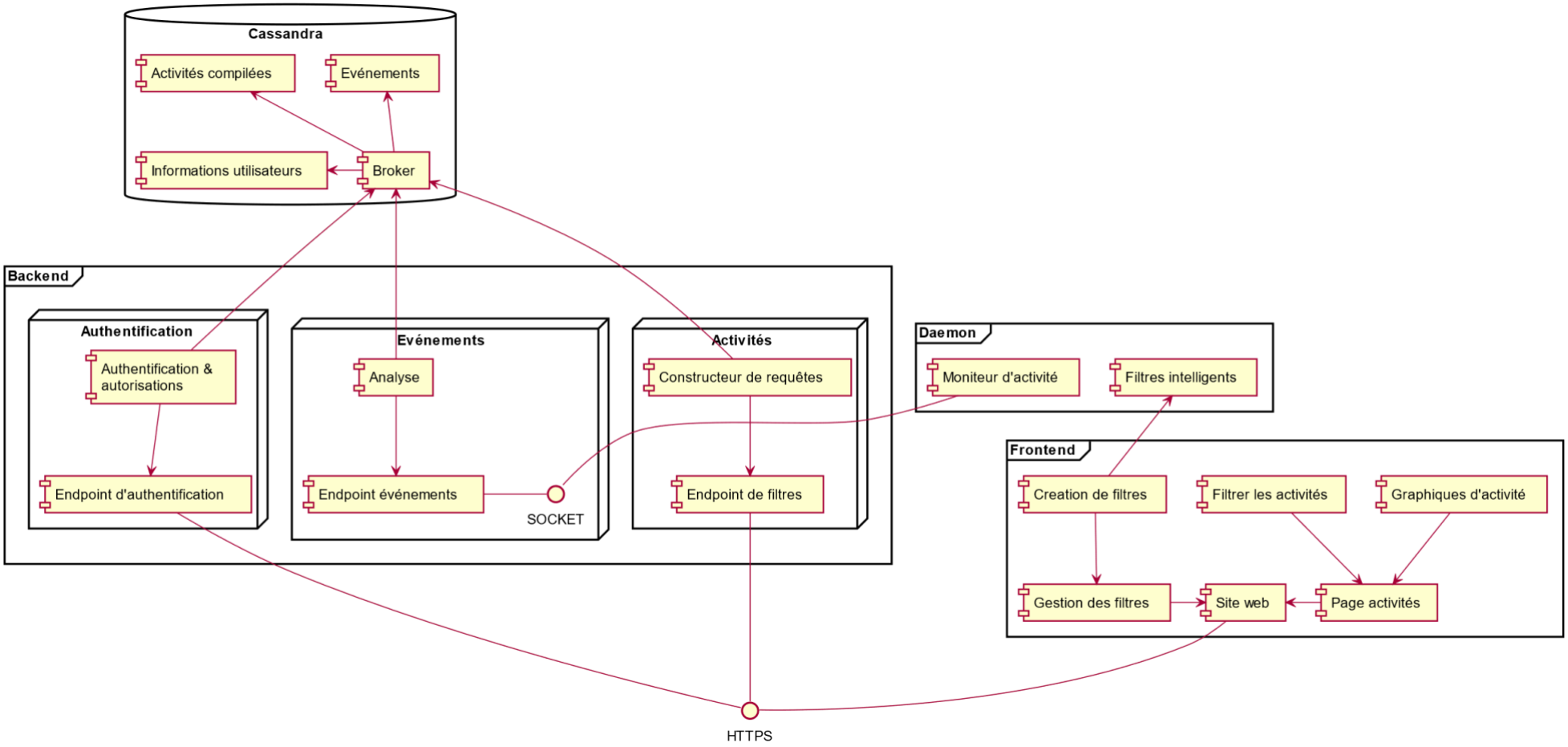

Figure 7 - Vue composants globale

Composants principaux

Chaque composant réside dans un Microservice séparé du reste de l'écosystème. Cette architecture permet de faire évoluer un module de manière simple, avec des répercussions et des effets de bord limités et contrôlés sur les autres modules.

Frontend

Le frontend est un site web offrant un contrôle sur le service Focus à l'utilisateur. La nécessité de pouvoir interagir rapidement avec les éléments des différentes pages nécessite que le frontend soit une application web monopage (plus connu sous le nom de *Single Page Application, SPA*) afin d'éviter d'attendre l'aller-retour de nombreuses requêtes. Cela permet aussi de réduire les coûts serveurs en déléguant la responsabilité de construire la page web au client web de l'utilisateur.

Cet aspect implique naturellement une API REST côté backend afin de faciliter les échanges grâce à des standards universels. Ce point sera détaillé plus loin dans cette partie.

Filtres : un filtre est une règle créée par l'utilisateur qui, une fois prise en charge par le Daemon, permettra de bloquer, temporiser, ou rediriger des nuisances numériques. L'interface de filtres du frontend communique avec le Daemon et le backend (pour stocker les règles) via requêtes HTTP vers une API REST hébergée dans un Microservice isolé.

A la manière de IFTTT ou Automator, l'utilisateur crée une règle réagissant à des événements : réception d'un e-mail, message ou notification, heure de la journée, certain logiciels/documents ouverts, etc, et détermine une(des) action(s) appropriée(s) à exécuter. Ex : « *si* je suis en train de travailler *alors* bloque les notifications Facebook. »

Ces règles sont créées dans le frontend avant d'être envoyées au Daemon et au Backend pour être enregistrées et synchronisées avec les autres appareils. Elles sont générées et stockées sous forme de JSON.

Activités : agissant comme la partie visible de l'iceberg, la page d'activité est un regroupement de toutes les informations dont dispose Focus sur l'organisation de la journée de l'utilisateur et sur l'utilisation de ses appareils. L'utilisateur dispose de multiples outils pour extraire l'information qui lui est importante sous forme de graphiques ou de texte.

La page d'activité est directement reliée au backend via HTTPS afin de recevoir les données les plus à jour sous forme de JSON. Le rafraichissement des données se fait toutes les 10 minutes ou au rechargement de la page. Augmenter cette fréquence serait peu utile vu la nature des informations rapportées.

Des filtres permettent de construire une requête qui sera envoyée au backend afin d'être traitée. Cette requête peut être générale (cas par défaut), auquel cas les informations relatives à la semaine courante seront renvoyées par le backend. Elle peut aussi demander une granularité plus élevée afin de fournir plus de détails à l'utilisateur. Les périodes de temps peuvent aussi être modifiées afin de fournir une vue plus ou moins détaillée selon les besoins.

Les filtres peuvent être agencés par différente catégories :

- Période de temps (jour, semaine, mois, année, total ou personnalisé)
- Type d'activité (travail, détente, etc)
- Applications (voir le temps d'utilisation d'une application)
- Projet (applications reliées à un projet spécifique)

Ces filtres sont amenés à évoluer en fonction des besoins et retours utilisateurs.

Daemon

APIs : Le daemon est un logiciel s'installant sur les ordinateurs de l'utilisateur. Une version du daemon existe par plateforme afin d'utiliser les API adéquates. Les APIs utilisées sont les suivantes.

- *Cocoa, Carbon et Quartz* sur Mac OS.
- *Win32 API*, bibliothèque bas niveau de Windows offrant une bonne rétrocompatibilité avec les versions précédentes de Windows.
- *Xlib*, permettant d'interagir avec le serveur X qui sert de serveur graphique à la grande majorité des distributions Linux. Cette librairie est aussi compatible avec le plus récent serveur *Wayland*.

Ce logiciel agit généralement de manière invisible aux yeux de l'utilisateur, ne demandant une action de sa part que pour l'authentification. D'autres contrôles simples sont également disponibles, comme la possibilité de mettre en pause le logiciel ou de changer de compte.

Evénements : Pour le cas d'utilisation principal, le daemon tourne en arrière-plan et récupère l'activité de l'utilisateur via les APIs susmentionnées. De nombreuses métriques sont récupérées afin de déterminer avec le maximum de précision l'utilisation réelle de la machine : titre des fenêtres, changements de fenêtre/onglet, temps passé par logiciel, onglets ouverts, fréquence de changement de contexte, nombre de notifications. Ces informations sont susceptibles d'être ajustées afin de refléter l'évolution de l'algorithme de traitement du backend.

Préanalyse : Une fois ces informations recueillies, elles passent par une première phase de préanalyse. Les informations sont d'abord nettoyées afin d'enlever les activités éphémères.

Ex : l'utilisateur doit passer de Word à ses mails, mais se trompe de fenêtre et donne le focus à Facebook qui était ouvert dans son navigateur. Le daemon aura enregistré cet événement, même s'il n'est pas pertinent (l'utilisateur a techniquement visité Facebook pendant un instant, mais sans l'intention de s'en servir). Cet événement peut donc être retiré sans risque afin de laisser la priorité aux applications réellement utilisées.

Sérialisation : Une fois nettoyées, les informations passent dans un sérialiseur qui convertira la donnée brute en l'insérant dans une « enveloppe » compatible avec les autres services Focus.

Cette enveloppe est une structure *Protobuf* définie globalement au travers des Microservices Focus. *Protobuf* est une bibliothèque de sérialisation et de *Remote Procedure Call* (RPC) open source développée majoritairement par Google. Au sein de Focus, elle garantit un modèle de données identique quelle que soit la plateforme, service ou langage de programmation utilisé (tant que ce dernier est supporté par *Protobuf*). L'autre avantage de *Protobuf* est la rapidité et légèreté des structures qui, une fois sérialisées au format binaire, offrent des performances supérieures au format JSON.

Cette enveloppe est composée d'un champ précisant le type d'événement sous la forme d'une énumération définie dans les structures *Protobuf*, permettant de savoir comment décoder le reste de l'enveloppe. Ce champ est suivi d'un message concernant les détails liés à cet événement. Par exemple, s'il s'agit d'un changement de fenêtre, le type d'événement ressemblera à USER_CHANGED_WINDOW, et permettra de déduire que le reste du message est composé :

- D'une chaîne de caractères *previous_window_name* qui correspond à la fenêtre sur laquelle l'utilisateur était avant d'en changer

- D'une chaine de caractères *current_window_name* qui correspond à la nouvelle fenêtre sélectionnée

- D'un entier *changed_after* qui sera le temps passé sur la fenêtre précédente.

Les types de message sont intimement liés aux événements que récupèrent le daemon, et sont donc naturellement amenés à changer en fonction des besoins.

Backend

« Backend » est un terme couvrant plusieurs Microservices qui agissent de concert pour fournir les capacités d'analyse avancées de Focus. Ce type d'architecture permet une excellente flexibilité en

garantissant une bonne interopérabilité entre les services. Il permet aussi de faire évoluer les différentes pièces tout en limitant et en contrôlant les effets de bords.

Authentification : gardien de Focus, le service d'authentification est garant des données et droits des utilisateurs. Il stocke les informations de compte des utilisateurs et permet de retrouver leurs données qui sont autrement stockées de manière anonyme, limitant les dégâts en cas de fuite de données.

Les données relatives à l'authentification sont stockées dans une base SQL, isolée des autres BDD.

Tous les services de Focus nécessitent un *JWT*, c'est-à-dire un jeton permettant d'authentifier l'utilisateur et de certifier ses droits. Ce dernier ne peut être crée et signé que par le serveur d'authentification grâce à une clé privée. Il contient l'identifiant unique de l'utilisateur, ses droits et une date d'expiration et peut être déchiffré et vérifié par n'importe quel service possédant le clé publique correspondante.

Si ce jeton n'est pas disponible, il est impossible de récupérer les informations de l'utilisateur. Les services doivent donc par conséquent renvoyer un code d'erreur jusqu'aux clients, afin que ces derniers demandent à l'utilisateur de s'authentifier.

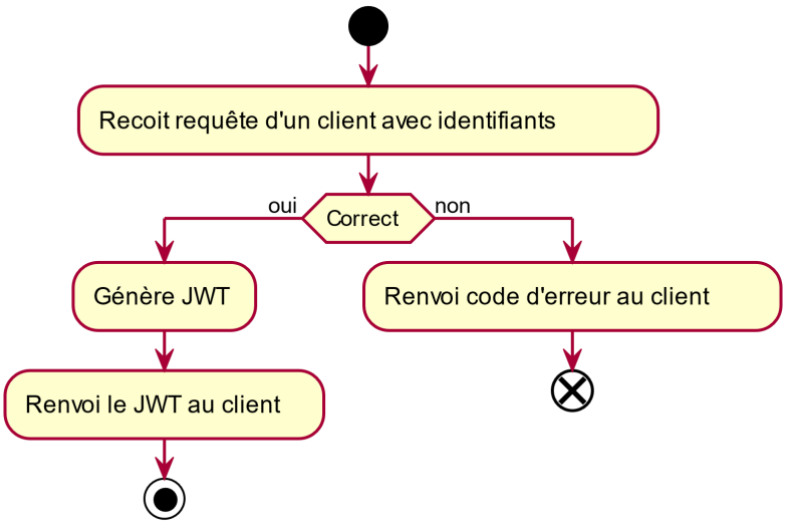

Figure 8 - Déroulement authentification au niveau d'un service

Afin de créer un JWT, le client doit communiquer avec le serveur d'authentification via HTTPS. Lors de la création du compte, le mot de passe initial est passé dans une fonction de hachage puis stocké dans la base de données d'authentification. Le mot de passe en clair est ensuite supprimé afin de limiter les dégâts en cas de fuite.

A chaque connexion, les informations de connexion (e-mail + mot de passe) sont envoyées de manière encryptée. Le serveur d'authentification applique la même fonction de hachage sur le

mot de passe et vérifie qu'il correspond bien avec le *hash* existant en base de données. Si les deux correspondent, un JWT est généré avec l'identifiant unique de l'utilisateur, des informations sur ses droits, et une date d'expiration. Cette date d'expiration est relativement courte (~10 minutes). Le jeton est ensuite ajouté à la base de données, associé à l'utilisateur qu'il authentifie.

Afin de ne pas devoir continuellement demander à l'utilisateur de se reconnecter, un jeton expiré peut être échangé contre un nouveau à condition que le jeton usé figure dans la base de données. Cette mesure permet à l'utilisateur d'invalider tout renouvellement de jeton s'il pense que son compte a été compromis.

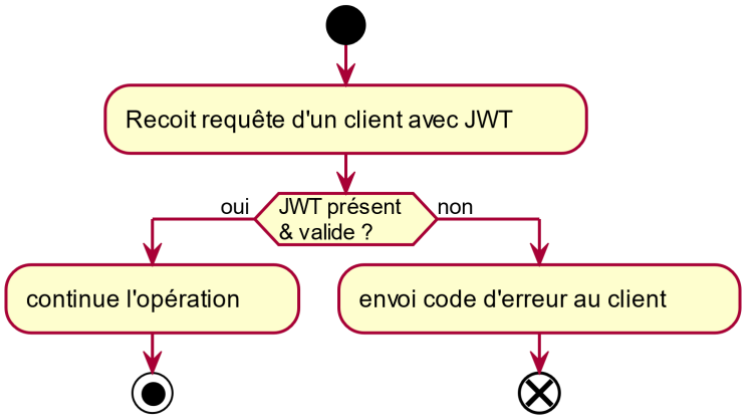

Figure 9 - Demande de JWT au service d'authentification

Persistance : les événements d'utilisation générés par les Daemon authentifiés sont communiqués au Backend via une connexion Socket TCP/IP. Le Microservice de persistance stockera :

1. Les données brutes, directement issues des données remontées par le(s) Deamon(s) de l'utilisateur, installé sur son(ses) appareil(s). Ces données n'ont pas vocation à être retenues indéfiniment, mais permettent de faire tampon entre la remonté de l'information et l'analyse effective des données par les modules de statistiques et d'analyse.

2. Les statistiques compilées, enregistrées dans un format structuré, à même d'être consommées par le frontend. De fait, les schémas des structures de données sont intimement liés à la nature des statistiques concernées. Chaque statistique distincte bénéficiera de sa structure de données propre et sa table associée.

3. Les données utilisateurs qui regroupent les informations le concernant ainsi que ses Daemon connus, accompagné des informations d'authentification.

4. Les états sérialisés des modules dormants. L'état des modules sujets à une inactivité prolongée est sérialisé et stocké. Quand un module est réveillé, son état est restauré depuis la version sérialisée présente en stockage persistant. Ainsi, nous optimisons l'usage de la

mémoire dans nos serveurs et mettons à disposition des modules une abstraction leur permettant de considérer qu'ils existent toujours en mémoire, réduisant leur complexité.

Une base de données Cassandra accueille la majeure partie des données. Capable de faibles latences et d'un débit important, Cassandra est à même de gérer un grand nombre de Daemon de manière simultanée. Les recherches sur les événements sont globalement limitées à la récupération sur une période donnée, les limitations de recherches de Cassandra ne sont donc pas impactantes.

Analyse : les événements recueillis sont trop détaillés pour permettre une utilisation directe, ils nécessitent une analyse qui a lieu dans le backend.

Du fait de la variété des statistiques que Focus a vocation à fournir sur l'usage que font les utilisateurs de leurs appareils, ainsi que la volonté de pouvoir fournir de nouvelles statistiques sans changer le modèle fondamental de leur génération, les choix suivants gouvernent le module d'analyse :

1. Asynchronicité de la génération. Les statistiques peuvent être interdépendantes (les statistiques les plus globales utilisent le principe de « top-down » en se basant sur d'autre statistiques), ainsi, une architecture asynchrone permet le calcul parallèle des statistiques indépendantes et, le cas échéant, une sérialisation implicite pour celles qui requièrent des étapes intermédiaires.

2. *Broadcast many-to-many*. Le module d'analyse se subdivise en une série finie de sous-modules, chacun ayant le rôle de calculer une statistique précise. Aussi, ces modules doivent pouvoir être déployés sur différentes machines. Le *Broadcast many-to-many* associé à une queue permet aux modules de s'inscrire aux topics (input) qui les intéressent et de *broadcaster* à nouveau les résultats des calculs.

3. Priorisation. La précédence de certains événements l'emporte sur d'autres, nous assurerons ainsi le traitement en temps-réel des événements sujets à des actions type blocage de notifications.

4. Sérialisation des états. Nous nommons les statistiques nécessitants des résultats de générations antécédentes (exemple, un module générant une moyenne, médiane, ou des quartiles) comme statistique *stateful*. Notre architecture prévoit que nos modules d'analyses vivent directement en mémoire sur nos serveurs et puisse être sérialisés et stockés en cas d'inactivité prolongée. Ainsi, un module recevant un nouvel élément disposera en mémoire des éléments nécessaires aux calculs des nouvelles valeurs.

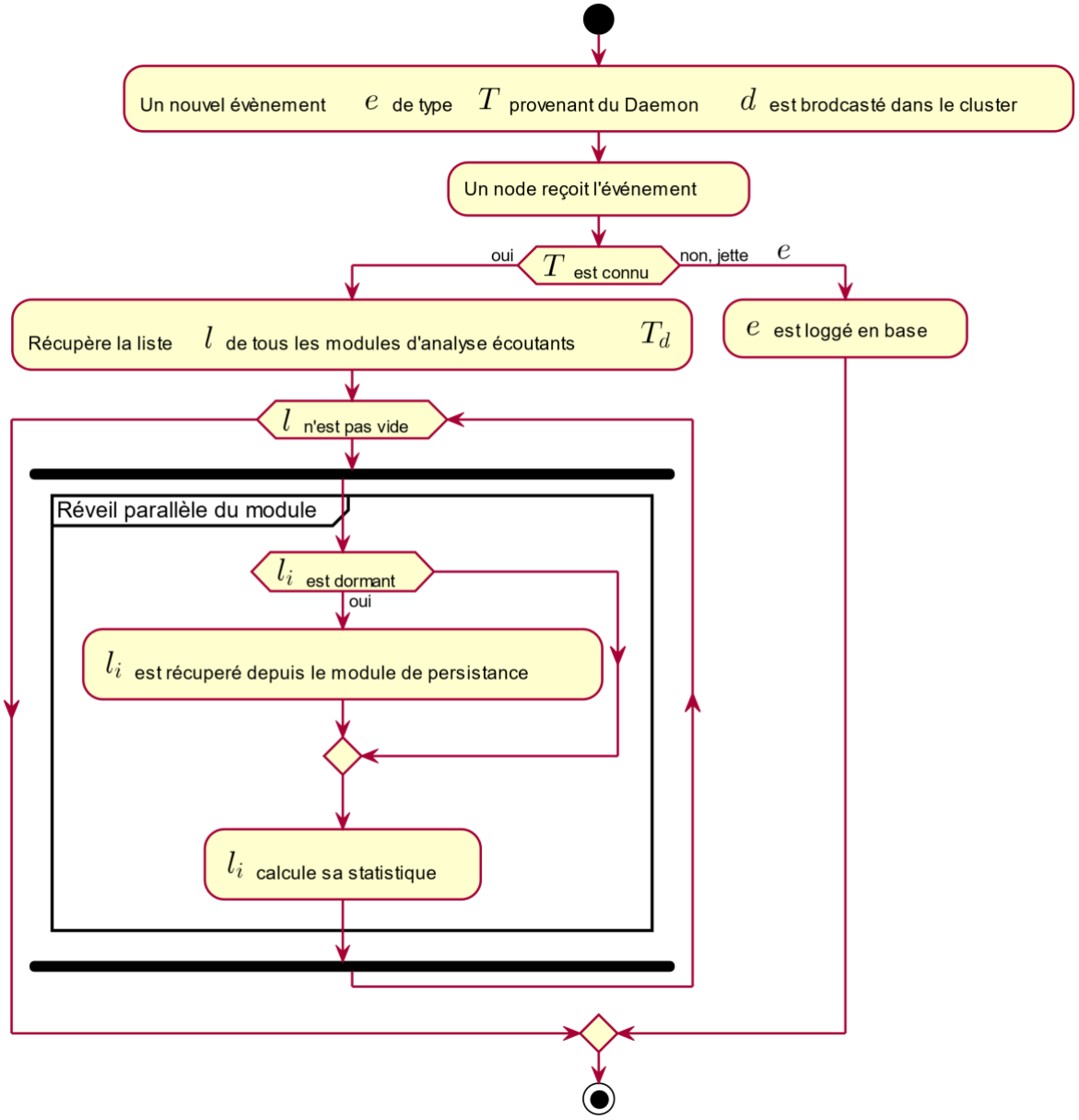

Figure 10 - Réveil des modules d'analyse

Le choix de cette architecture permet la mise à l'échelle sur un nombre arbitraire de machines ainsi que de mettre à disposition (1) de nouveaux types d'événements suite à une mise à jour des Daemons, (2) de mettre en place de nouveaux modules d'analyses sans conflits avec les autres et (3) de décorréler le nombre d'événements transmis par les Daemons avec la génération de statistiques, la queue jouant le rôle de tampon. De même, cette architecture est hautement parallèle par nature, puisque les modules réagissent à des messages et non pas à une série préprogrammée d'actions.

En dernière instance, chaque module d'analyse concernant un type de statistique finale (*i.e.* une statistique non-intermédiaire, contingente à la création d'autres statistiques) broadcastera un dernier message à destination des modules de persistance, qui insèreront la statistique N+1 dans la base de données appropriée.

Les filtres définis par l'utilisateur sont conceptuellement équivalent aux modules de génération de statistiques en cela qu'ils écoutent un certain nombre de messages et produisent un résultat. Celui-ci néanmoins diffère en nature. Il ne s'agit pas d'une nouvelle statistique mais d'une action, transmise au module de communication avec le Daemon pour que celui-ci effectue l'action demandée (ex. bloquer les notifications venant de telle application, si applicable).

Via cette approche, nous mettons en place une architecture à même de

1. Maximiser notre capacité à produire des statistiques le plus rapidement possible

2. Mettre à l'échelle notre cluster sans friction, en répondant linéairement à la bande passante d'événement entrante

3. Produire de manière agile de nouveaux modules de statistiques en minimisant les effets de réactions en chaine sur les autres

4. La séparation des éléments fonctionnels, chaque module de statistique étant clairement identifié et autonome

5. Créer une sémantique puissante pour la définition de règles pour nos utilisateurs. Chaque règle étant indépendante des autres et pouvant se connecter sans effort supplémentaire à toutes les statistiques indépendamment générées par les autres modules

Vue Processus

Communication

Comme précisé auparavant, Focus repose sur une architecture de Microservices et d'un système d'acteurs. Ici, un service désigne un bloc logique de l'application Focus : serveur d'authentification, module d'analyse, persistance, bases de données, etc. Les acteurs se situent à un niveau plus bas, et représentent chacun une entité réelle. Nous aurons donc par exemple une instance d'acteur *DaemonActor* pour chaque Daemon connecté au serveur. Les services et acteurs étant isolés et ignorants du reste du réseau, ils doivent communiquer par un serveur central.

Nous utilisons RabbitMQ pour gérer cette communication. Cette bibliothèque de *Message Broker* est l'une des plus matures & évolutives du marché et est utilisée par de grandes entreprises en production. Elle assure que l'échange de messages se fasse de manière consistante et sécurisée grâce à un modèle dit *pub-sub*.

Chaque service démarre en se connectant au *Message Broker*, puis s'inscrit à différents événements. Dès qu'un événement correspondant est reçu par RabbitMQ, tous les services qui s'y sont inscrit sont notifiés. Les messages peuvent inclure des données pour faciliter les échanges. Par exemple, les modules d'analyse peuvent être notifiés de l'arrivée de nouvelles données sans avoir à connaître quoi que ce soit du réseau, en étant juste connectés à RabbitMQ.

Authentification

Le serveur d'authentification est un service à part entière, possédant sa propre base de données. Son rôle se limite à la création, enregistrement & gestion des jetons d'authentification, ainsi qu'à la création & gestion des comptes utilisateurs.

Toutes les informations relatives aux comptes utilisateurs sont stockées dans une base de données spécialisée. Seul le serveur d'authentification peut y accéder afin de limiter le risque de fuites.

Création de compte

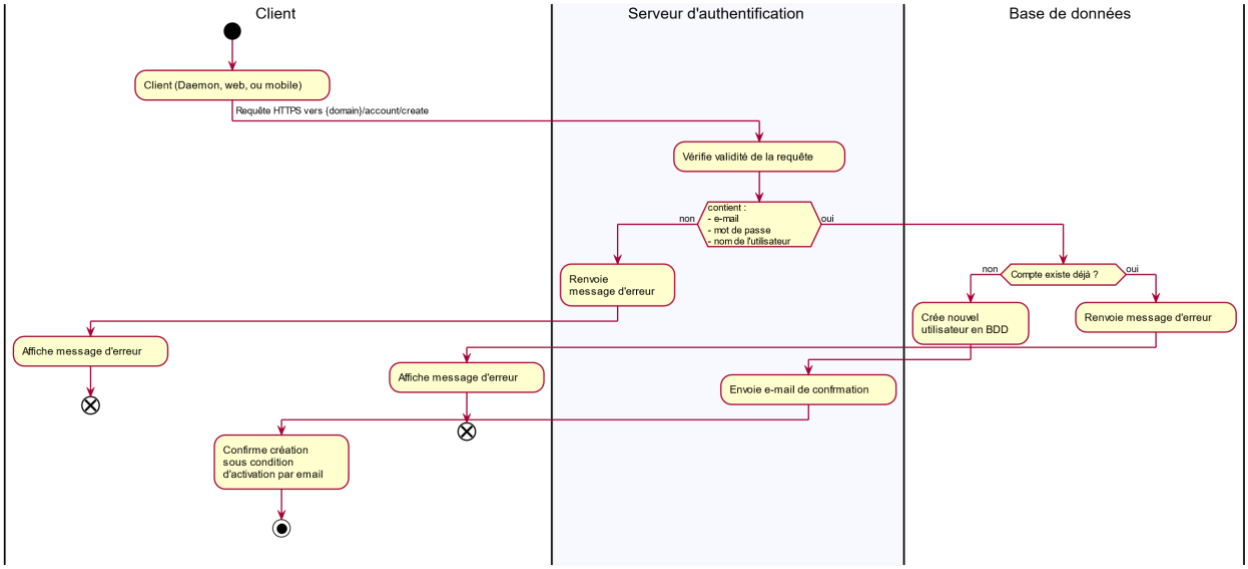

Figure 11 - Diagramme d'activité de création de compte

Le modèle de création de compte est simple, des garde fous basiques sont mis en place pour assurer la validité de la requête qui, si correcte, résulte en la création d'un nouvel utilisateur.

Il est important de noter que la base de données dans la figure ci-dessus désigne la BDD d'authentification, qui siège dans un service isolé des autres.

Authentification de l'utilisateur

L'authentification est nécessaire à certaines étapes du processus : lors de la création d'un socket entre un Daemon et le Backend, pour accéder aux informations compilées et pour accéder/modifier les préférences de compte.

Tous les services nécessitent un *JSON Web Token* (JWT) pour garantir l'identité de l'utilisateur faisant usage du service. Ce JWT ne peut être généré que par le serveur d'authentification.

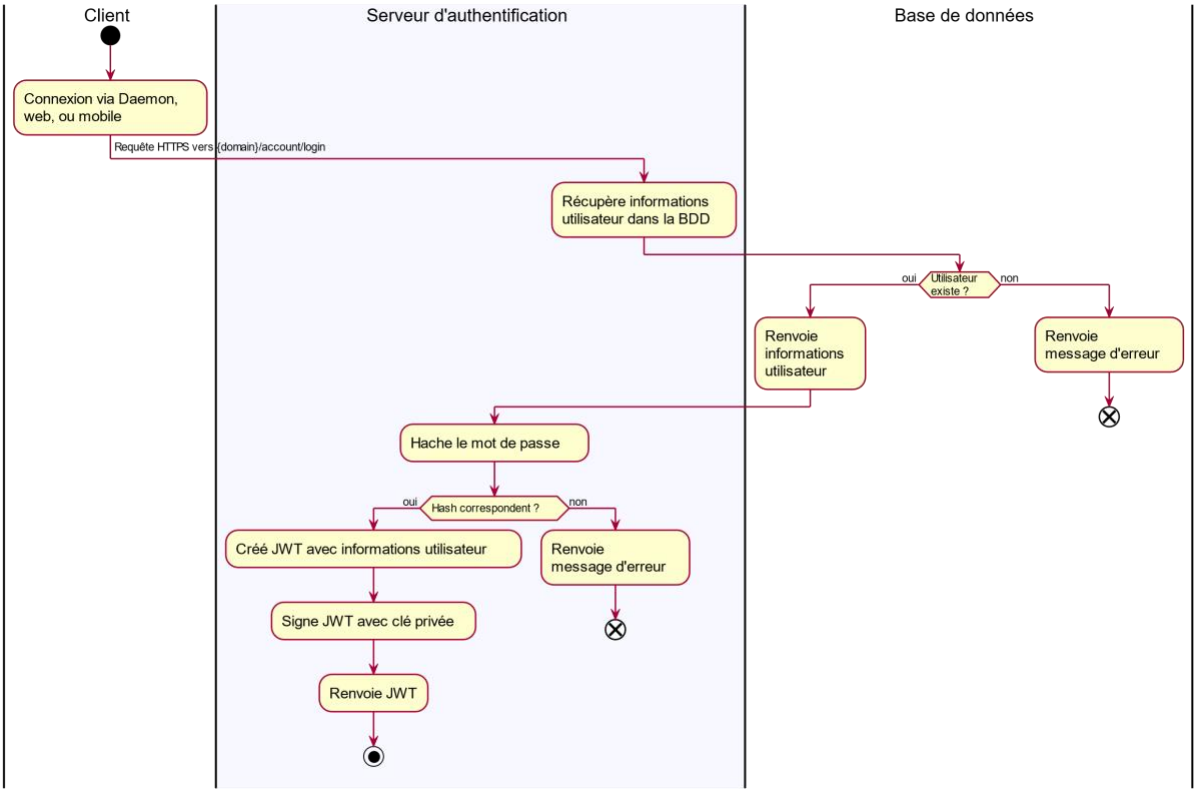

Figure 12 - Création d'un JWT

Le JWT doit alors être attaché à toutes les requêtes nécessitant une authentification. Il est aussi programmé pour expirer au bout d'un période prédéterminée et doit donc être échangé contre un nouveau JWT valide. Ce mécanisme permet de donner plus de contrôle à l'utilisateur quant à l'accès aux services en son nom.

Monitoring

Le monitoring de l'activité de l'utilisateur est la première moitié du couple de fonctionnalités clés de Focus. L'acteur principale en est le Daemon, qui interagit avec les APIs systèmes directement afin d'avoir le maximum de détails quant à l'utilisation de la machine. Fenêtres ouvertes, frappes clavier, sites ouverts, et autres événements sont récupérés, nettoyés et envoyés au backend pour une analyse plus poussée.

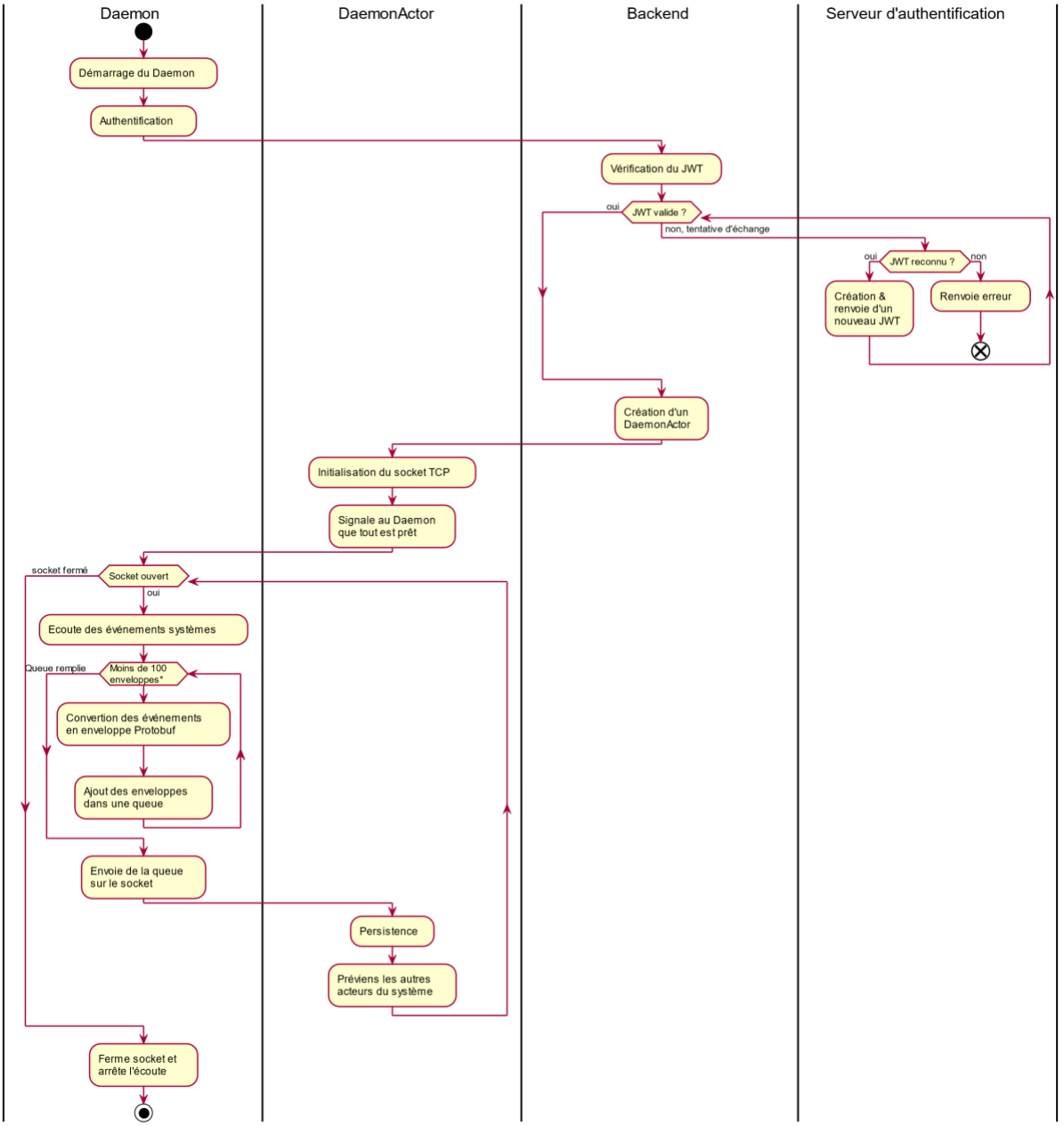

Figure 13 - Processus de monitoring du Daemon

**Moins de 100 enveloppes* : ce nombre est aujourd'hui arbitraire, et sera sujet à évolution en fonction des performances mesurées.

Nettoyage des événements

Par défaut, tous les événements sont enregistrés. Chaque changement de fenêtre, chaque clic, chaque frappe clavier est enregistré, résultant en une grande quantité de données. Cependant, seule une portion de ces informations est pertinente, le reste n'est que bruit.

Par exemple, dans le cas où l'utilisateur fait défiler ses fenêtres pour passer d'un document Word à sa boîte mail, il peut donner le focus à son navigateur l'espace d'un instant. Pourtant, aucune attention n'a été portée sur le navigateur, par conséquent, il n'est pas approprié de dire que l'utilisateur utilisait son navigateur à ce moment. Les événements très courts et n'enregistrant aucune activité particulière (clic, frappe dans la fenêtre) peuvent être supprimés sans risques.

Analyse

L'analyse est l'étape qui transforme les données récupérées par le monitoring du Daemon et la transforme afin de la rendre compréhensible et utilisable. Les détails intrinsèques à ces analyses sont destinés à changer au fur et à mesure des expérimentations, et des retours utilisateurs et analytiques.

A la manière d'un réseau de neurone grandement simplifié, le service d'analyse se décline en une multitude d'acteurs indépendants qui communiquent entre eux via message relayés par RabbitMQ. Les acteurs sont développés pour une tâche précise & élémentaire et se chaînent afin de produire un résultat final plus complexe. Les acteurs situés au plus bas niveau se chargent des opérations les plus basiques, tandis que les ceux opérants à un niveau plus élevé utilisent ces résultats pour générer des informations plus complexes et riches. Ces opérations s'effectuent de manière asynchrone et peuvent être déployés sur plusieurs machines en fonction des besoins à un temps T.

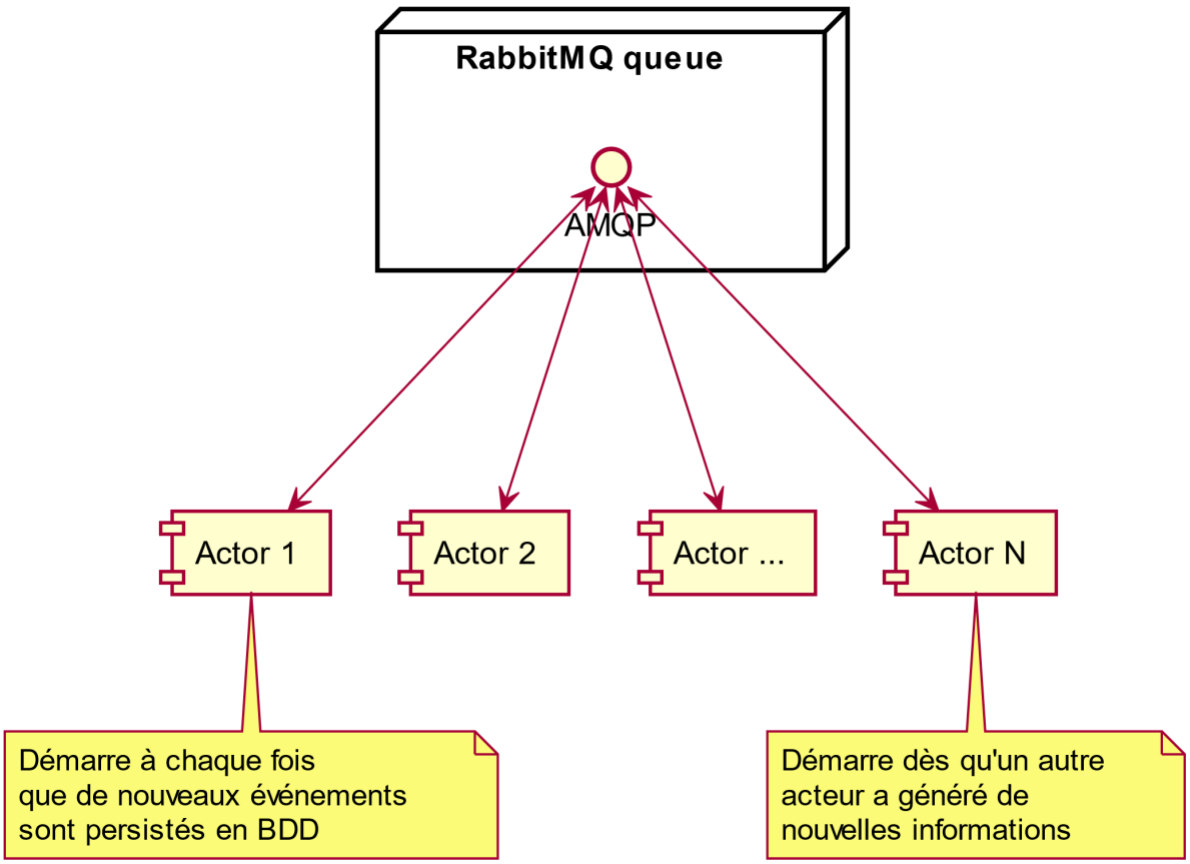

Figure 14 - Intéraction des différents acteurs d'analyse

Implémentation

Frontend

Le choix de langage de programmation est limité pour le frontend web, nous devons utiliser du HTML5, CSS3 et JavaScript. Nous faisons usage des derniers standards ECMAscript, ES8 afin de bénéficier des versions les plus à jour du langage. Ce dernier n'étant cependant pas supporté par tous les navigateurs, nous devons le transpiler en ES5 grâce à divers outils.

Nous utilisons un modèle MVC standard, avec une hiérarchisation en composants isolés au maximum. Chaque composant doit pouvoir être réutilisable par d'autres composants. La donnée chargée dans le frontend est gérée par un *store* unique et global. Chaque composant réagit aux changements du *store*, et ne peux interagir qu'avec ce dernier ou avec son propre état qui est isolé du reste. Un composant ne peut pas directement influencer l'état d'un autre, ceci afin d'éviter les régressions et effets de bord dans le futur.

Backend

Nous avons mentionné tout au long du document l'utilisation de Microservices et de systèmes d'acteurs, qui représentent le squelette de l'architecture de Focus. Ces choix techniques nous offrent de nombreux avantages qui siéent parfaitement avec les besoins du projet : scalabilité, interopérabilité, performances et coûts.

Les librairies principales sont détaillées ci-dessous.

C++ Actor Framework. Plusieurs librairies d'actor system développées en C++ coexistes, chacune offrant avantages et inconvénients. C++ Actor Framework (CAF) est développé en s'inspirant du langage de programmation Erlang, en s'assurant de garder des performances et une empreinte mémoire excellentes.

RabbitMQ est notre service de messagerie inter-services. Ce logiciel open-source offre des fonctionnalités de *pub-sub*, de persitance des messages et de bonnes performances. C'est l'une des solution de Message Queue les plus utilisées sur le marché.

Notre langage de programmation est le C++ (version 17). Ce langage polyvalent, mature et performant est maîtrisé par tous les membres de l'équipe et semble un choix adapté au projet.

Certains services du projet ne sont toutefois pas adaptés à ce choix de langage. Par exemple, les APIs système de MacOS, ou le frontend web et mobile nécessitent chacun un langage précis. Il s'agit donc de limiter au maximum la friction due à la différence entre ces langages.

Pour ce faire, nous devons adopter des outils de communication agnostiques au langage et à la plateforme. En plus de RabbitMQ précédemment mentionné, nous utilisons Protobuf. Développé par Google, Protobuf nous permet de communiquer de manière performante pour envoyer les événements des daemons, en sérialisant les enveloppes en binaire. Protobuf permet aussi de faire du *Remote Procedure Call*, qui consiste à appeler des fonctions sur un service différent, qu'il soit sur une autre machine, ou sur un service développé dans un langage différent.

Daemon

Le développement du Daemon est, pour sa majorité, conditionné par la plateforme sur laquelle il est fait. Une première base commune est développée en C++, langage supporté par Windows, Linux et MacOS. Une abstraction est établie pour les APIs système en C++ et implémentée sur chaque plateforme.

Pour la communication avec le backend, la bibliothèque *NanoMsg* est utilisée pour sa simplicité d'utilisation, robustesse et ses performances. Les informations sont sérialisées avec *Protobuf*.

Vue Déploiement

Outre les daemons qui nécessitent d'être développés sur les plateformes majeures, le reste des services tourne dans un environnement dit *containerisé*. Tout service réside dans un container Docker, solution mature et répandue.

Nous utilisons la combinaison de CoreOS et de Kubernetes afin de gérer et orchestrer les différents services.

Nos services sont hébergés chez Amazon Web Services. Les géants de l'IAAS proposent des outils de monitoring, de configuration et déploiement performants et adaptés à nos besoins. Nous pouvons aussi faire usage des *Spots Instances* proposées par AWS, qui sont des instances de machines virtuelles à moindre coût, mais sans garantie de durabilité. L'instance peut être détruite à tout moment. Cette offre est parfaitement adaptée pour nos calculs d'analyse qui peuvent se faire de manière asynchrone, réduisant ainsi nos coûts d'infrastructure.

CoreOS

CoreOS est un système d'opération basé sur Linux créé pour la gestion de containers. Des outils de distribution sont inclus de base, tels que **etcd** (partage de configuration synchronisé sur plusieurs machines). CoreOS a récemment adopté Kubernetes comme remplacement à **Fleet** pour gérer les différents containers de manière automatisée ou manuelle.

Kubernetes

Kubernetes est une solution de containerisation développée principalement par Google, puis rendue open-source. Elle permet de déployer ou éteindre des containers en fonction des besoins, ou de relancer les services en cas d'erreur. D'autres fonctionnalités seront la bienvenue dans le

futur, comme le Load Balancing qui permettra de répartir les charges de travail en fonction des ressources disponibles.

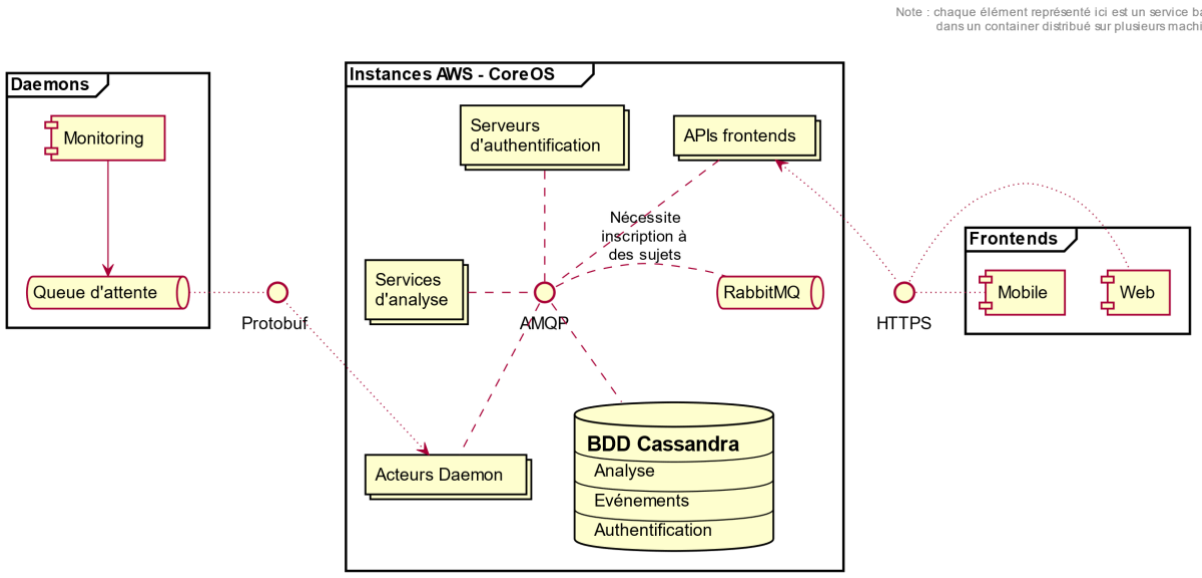

Figure 15 - Diagramme de deploiement global.

Vue donnée

Nous traiterons dans cette partie du data model de Focus. Le cœur de métier de notre projet étant la collecte de données, une attention toute particulière doit être portée à cette partie afin de bien comprendre les interactions entre nos différents composants (daemon, backend et frontend). Les informations données dans cette partie peuvent être soumises à quelques changements mineurs en fonction des besoins qui pourraient être rencontrés lors du développement, notamment quant à la gestion des événements sur le Daemon.

Interaction Daemon/Backend

Le daemon est la source de données qui propulse Focus. A chaque événement de l'utilisateur (ouverture d'une fenêtre, changement d'onglet, clic de souris, frappe de clavier, etc.) une action est générée dans le daemon et une enveloppe Protobuf est créée de la manière suivante :

- On ajoute le UUID du daemon à l'enveloppe
- On indique le type de donnée qui va être ajoutée (*WindowsContextPayload, MouseEventPayload, KeyEventPayload*) en fonction de la nature de l'événement
- Enfin on incorpore la donnée brute à notre enveloppe

Le schéma ci-dessous fourni une représentation de notre modèle de donnée pour la communication des événements entre les daemons et le backend.

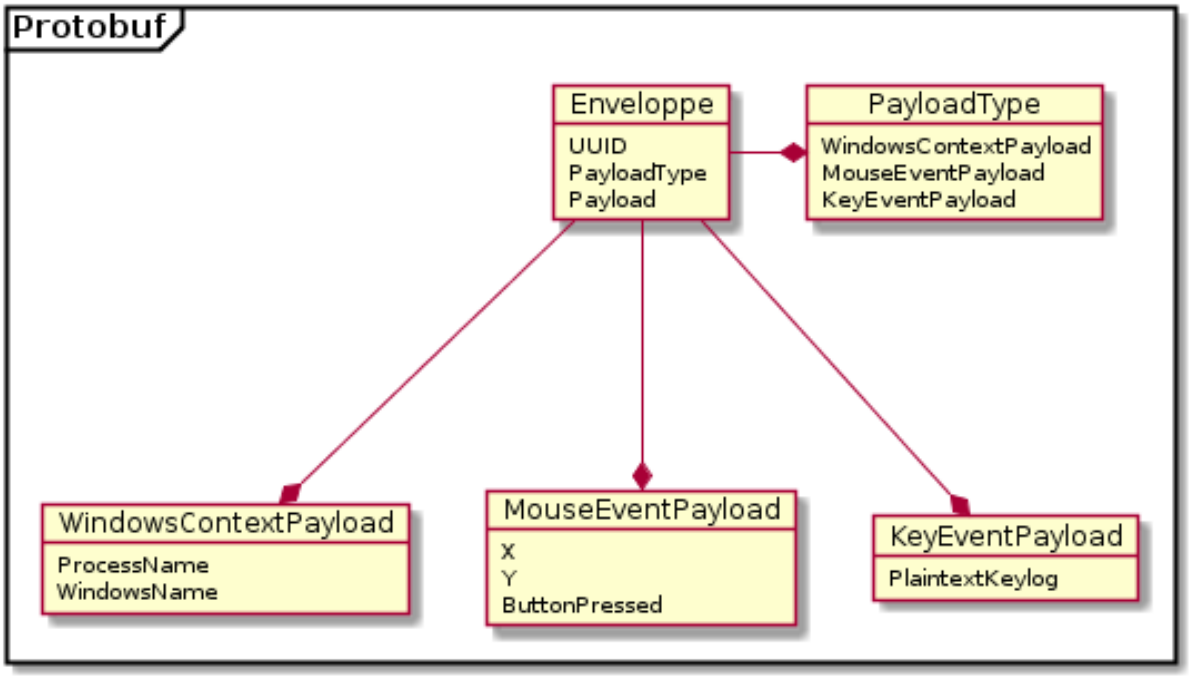

Figure 16 - Enveloppe Protobuf

Cette enveloppe sera ensuite sérialisée en binaire et envoyée au travers d'un socket jusqu'au backend qui va la désérialiser et en extraire les informations.

Protobuf étant une bibliothèque de sérialisation *cross-language* et *cross-platform*, elle nous fournit une très bonne abstraction de haut niveau sur la sérialisation et nous permet d'évoluer plus facilement dans le futur si des changements d'architecture ou de langage venaient à s'imposer.

Stockage des évènements utilisateurs dans le Backend

Lors de la réception d'un événement par le Backend, celui-ci va le stocker dans une base de données, afin de l'enregistrer et de pouvoir le traiter et l'analyser par la suite. Un model NoSQL a été adopté pour accueillir, classifier et organiser ces informations.

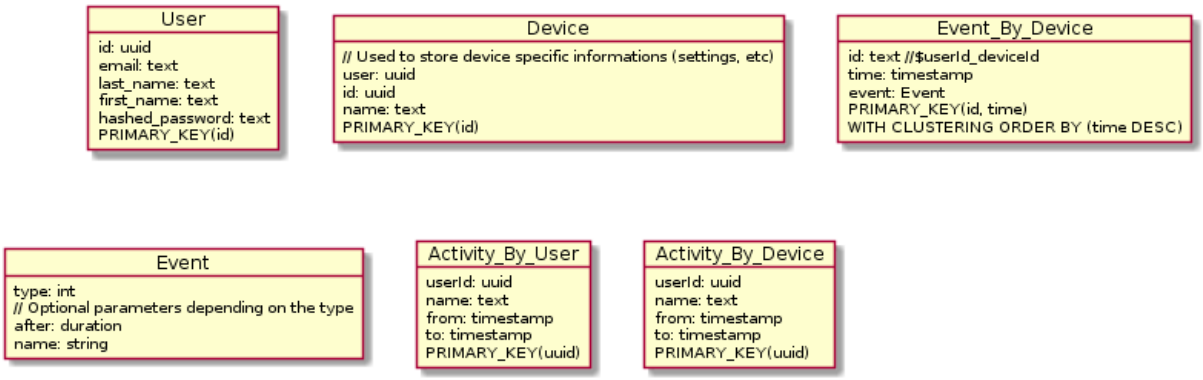

Figure 17 - Data Model Backend

Dans notre architecture, un utilisateur peut avoir plusieurs périphériques, mais chacun de ces périphériques ne peut être relié qu'à un seul utilisateur. Par conséquent nous enregistrons les événements par périphérique et non pas par utilisateur. La table *Event_By_Device* (voir figure ci-dessus) nous sert à stocker chaque événement renvoyé par le daemon. L'événement est démarqué grâce à l'identifiant unique de l'utilisateur, celui de son périphérique et d'un *TimeStamp*.

Lorsque le Backend possède suffisamment d'événements pour calculer une activité, il va analyser les listes d'événements en fonction des différents utilisateurs et de leurs périphériques respectifs (ex : Jean ouvre Word à 09h30, Jean ferme Word à 11h00, alors Jean a utilisé Word pendant 1h30).

Interaction Backend/Frontend

Il est de la responsabilité du frontend de mettre à disposition les rapports d'activité de l'utilisateur. En effet, c'est le frontend qui est l'interface entre nos services et nos utilisateurs.

Lorsqu'un utilisateur se connecte sur son compte Focus via notre interface web ou nos applications mobiles, ce dernier va interroger une REST API afin d'obtenir les dernières activités de l'utilisateur et de ses périphériques. La REST API va ensuite récupérer les données stockées dans la base de données puis va les sérialiser dans un format nommé JSON afin d'être renvoyé au frontend qui pourra interpréter la donnée correctement.

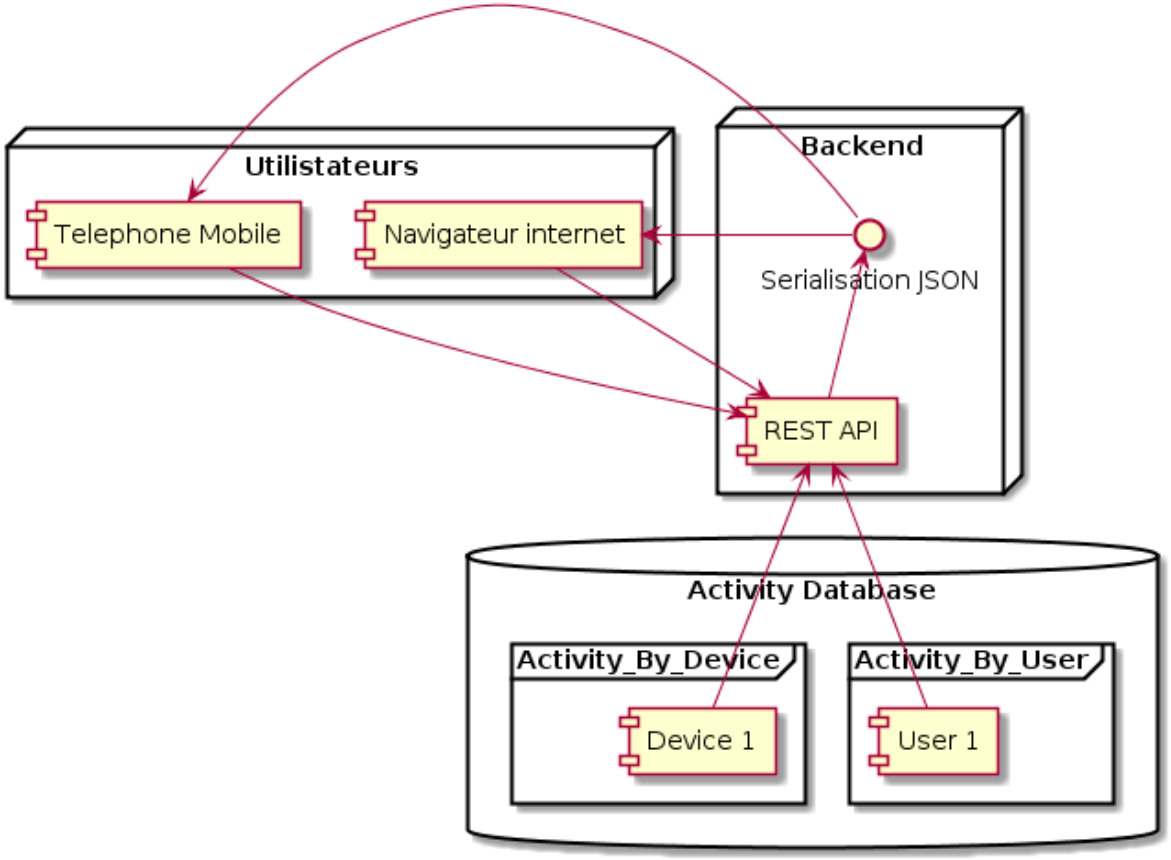

Figure 18 - Communication entre le backend et le frontend

Qualité

La qualité de notre produit est assurée par un développement continu en TDD (*Test Driven Developement*). C'est-à-dire que pour chaque fonctionnalité de notre produit nous écrirons d'abord une série de tests unitaires avant de débuter le développement à proprement parler. Ceci nous permet de nous assurer que le code que nous allons produire répond à une exigence de qualité suffisante. De plus cela nous permet lors de la poursuite du développement de nous assurer qu'il n'y a aucune régression.

Le Framework de test unitaire « Google Test » a été intégré au projet dès son lancement, ce dernier nous donne des outils d'analyse relativement poussés afin de pouvoir surveiller nos tests.

De plus, notre code partagé est déployé sur deux plateformes de tests unitaires à chaque *commit* (Appveyor et TravisCI, voir figures ci-dessous).

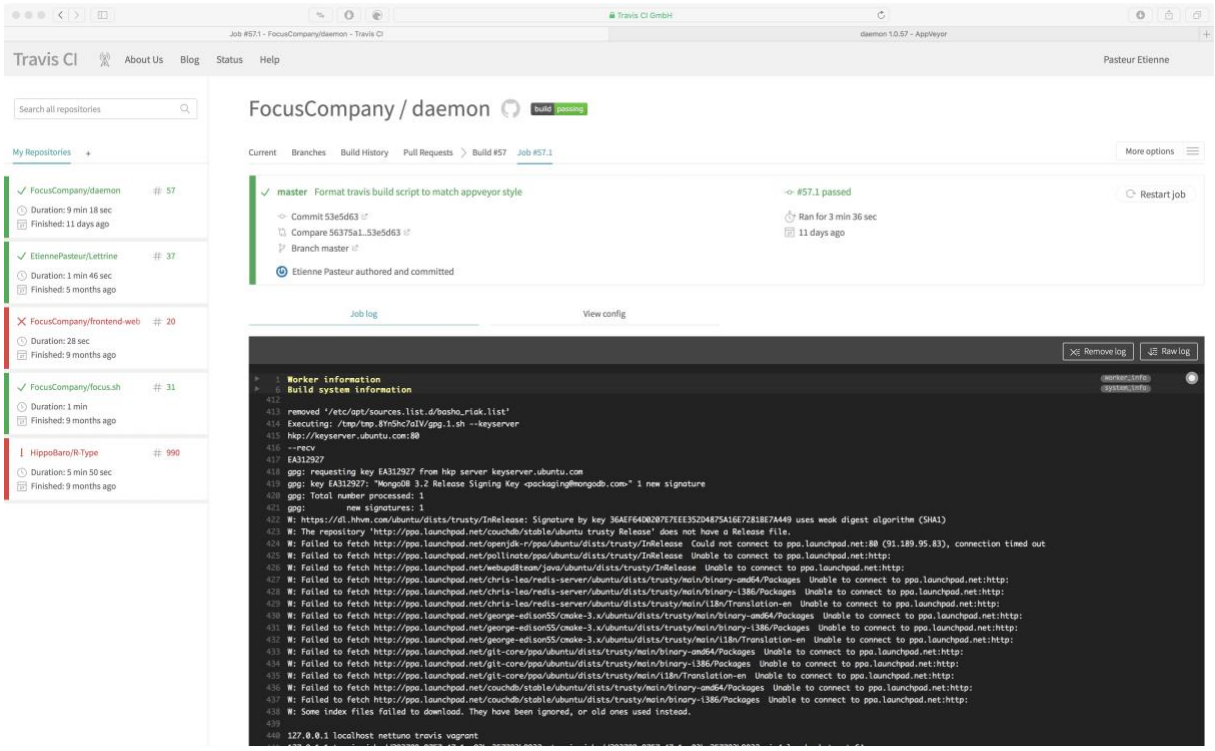

Figure 19 - Déploiement sur TravisCI

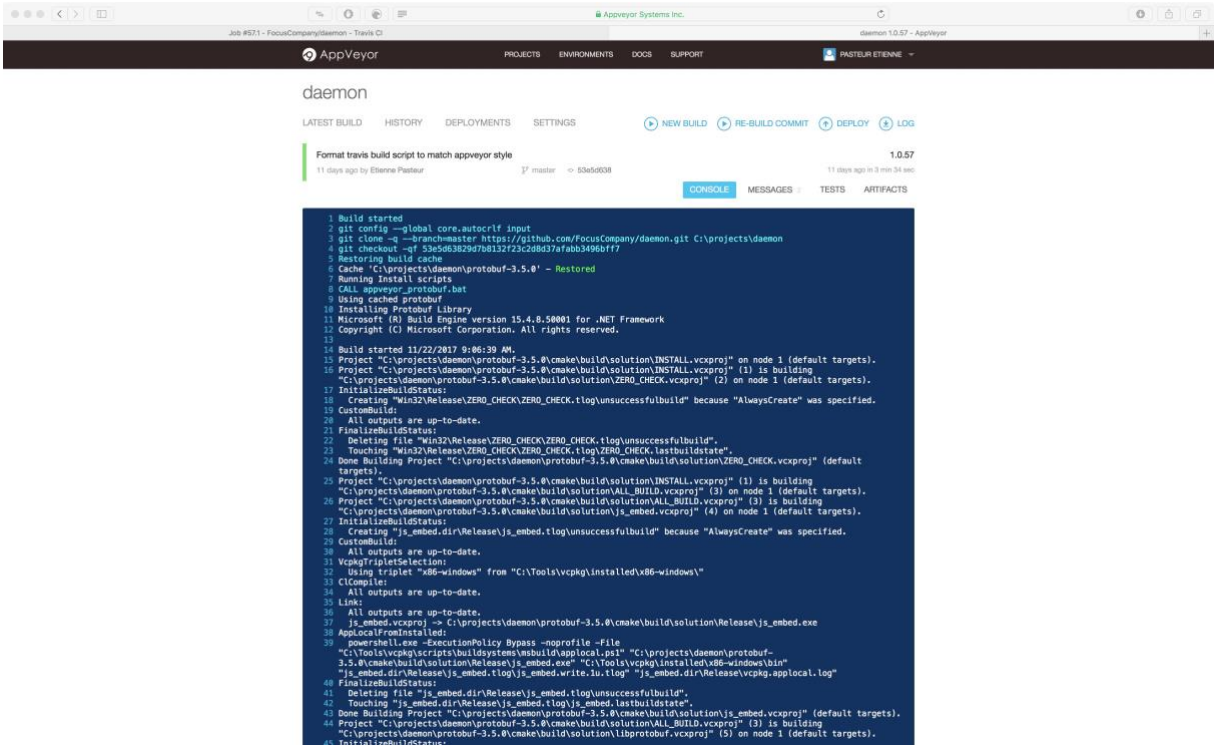

Figure 20 - Déploiement sur Appveyor

Grâce au déploiement continu sur ces deux plateformes, nous sommes en mesure de nous assurer que le code que nous partageons au sein de l'équipe sur GitHub (plateforme de partage de code utilisant l'outil de versionnage de fichiers Git) répond pleinement à toutes nos exigences en termes de tests unitaires. Cela nous permet de toujours avoir une base de code saine qui fonctionne sur toutes les plateformes.