COMPLEMENT

Ayant fini la réalisation du pipeline plutôt que prévu, j’ai pu faire d’autre tâches dans l’entreprise. La partie qui suit est un aperçu global et chorologique des tâches supplémentaires qui m’ont été conféré.

1. BASCAC – Blu Age (J’ai oublié la suite).

BASCAC est une petite application web qui permet de simplifier la gestion de la facturation de Velocity au client. Celle-ci s’appuis sur la base de données client (identique à la base de données qu’utilise le pipeline pour récupérer les ACL).

L’application permet entre autres, d’ajouter/retirer la permission à un client d’utilisé Velocity, ou encore de géré le pourcentage de réduction accorder au client. En conséquence, son accès doit être limité car elle gère des données de facturation.

L’objectif de cette tâche est de mettre en ligne BASCAC de manière sécurisée, en limitant son accès par un mot de passe. J’ai utilisé l’IDE Visual Studio Code pour y ajouter mes modifications.

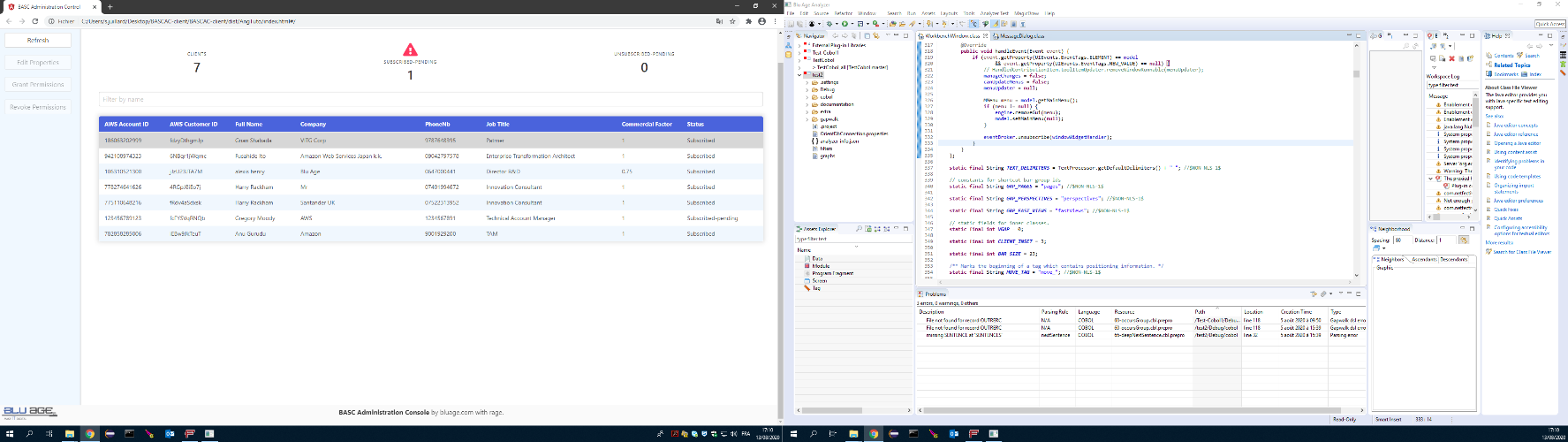
L’outils qui m’as été préconisé est AWS Cogito, un service d’amazone, pour les applications web, qui propose de gérer les mots de passe.

L’application a été réalisé avec Angular, soit codé en TypeScript, html et css. L’application été nativement une “single-page application” ou SPA. J’ai dû rajouter à cette application un mécanisme de routage afin d’y ajouter une page d’authentification.

Une fois cela terminer, j’ai essayé d’intégrer cogito à ma page d’authentification. C’est à cette étape que j’ai pris connaissance de Amplify. Un autre service d’amazone qui permet de mettre en ligne des applications. Pour la mise en place de Cognito, son utilisation été nécessaire.

En me renseignant plus sur cet outil, je me suis aperçu que je pouvais déployer BASCAC, après l’avoir compilé, de manière non-programmatique sur la console graphique d’Amazone. Puis une fois cela fait, ajouter la gestion des mots de passe, qui apparait comme un pop-up en haut de la page. (Cf. image)

Au final, je suis revenu sur mon travail d’ajout de page d’authentification et d’utilisation de Cognito et j’ai opté pour un déploiement et une gestion des mots de passe non-programmatique, à l’aide d’amazone.

N.B : Sur demande de son créateur, j’ai également modifié légèrement le style de la page en ajoutant du contenu aux feuilles de styles de l’application.

2 – Exportation de la Vue Problème de Blu Age Analyzer.

Analyzer est un produit de Blu Age de la V7 <ALORS LA J’AI BESOIN D’EXPLIQUATION POUR SON USAGE>.

Lorsqu’un client rencontre des problèmes avec son code, il peut contacter Blu Age afin de recevoir de l’aide. On lui demander alors le contenu de l’onglet « problème », l’onglet dans lesquels est affiché les erreurs et les warnings issue de l’a compilations du/des projets en cours.

L’objectif de cette tâche est d’ajouter, dans l’onglet problème une option pour exporter le contenu de la vue au format .csv (tableurs).

Blu age Analyser est une IDE développé sur la base d’Eclipse. Elle possède à la fois des onglet, vues et fonctionnalités développées par Blu Age et d’autre issue de Eclipse.

En xml, on peut ajouter des éléments graphiques à Analyzer, et les liés avec des handlers, commandé en java.

La difficulté de cette tâche réside dans l’adaptation de code interne à Eclipse. L’onglet « problèmes » d’Analyzer, est une vue dont le code est interne et il est donc difficile d’en récupérer les éléments. Pour cette raison, il n’est pas possible de récupérer le contenu de la vue problème à proprement parlé. Pour récupérer ce contenu, il a fallu aller à la source de ce contenu : On va récupérer les informations « à la source », soit dans les markers d’un projet. Un marker est un ensemble de donnée généré dans notre cas à la racine du projet dans lesquels les informations relatives à celui-ci son stocker.

C’est en récupérant ces informations que j’ai pu reconstituer la vue problème, puis l’exporter sous le format csv. Après son implémentation, j’ai pris connaissance d’une option similaire, qui avait déjà été implémenter, mais qui avais un usage différent. Cette option proposait d’exporter, à la racine du projet, dans plusieurs formats possibles, les problèmes relatifs à un projet en faisant une clique droite sur celui-ci dans la vue navigation.

J’ai donc dans un deuxième temps, adapter ma solution pour quelle s’intègre au travail déjà effectuer. Ainsi, sans retirer l’option déjà implémenté, j’ai ajouté à la vue problème une option qui propose d’exporter la vue problème (problèmes de tous les projets ouvert), dans plusieurs formats différents, dont le csv, et de l’enregistre par un menu contextuel en choisissant le nom et l’emplacement du rapport d’erreurs.

Une fois la tâche terminer, j’ai pris connaissance des protocoles de Blu Age afin de manager le suivit des commit. Avec l’aide de Oliver <Nom de famille> j’ai revu l’écriture de mon code, j’ai rempli mon message de commit avec un identifiant qui correspondais à ma tâche, puis on a rempli ensemble cette fiche pour présente l’option implémenté.

Conclusion :

Ce stage ma permis de prendre en compétence, et ce dans diverse domaine : devOps, devWeb et dev logiciel. J’ai codé dans plusieurs langages différents, utilisé et tester divers outils. Cette expérience m’as permis de développer ma capacité d’adaptation et ma flexibilité.

J’ai dû moi-même concevoir des solutions, chercher par moi-même les outils qui pourrais me servir et faire le choix des plus adéquats. Pour la tâche d’implémentation d’une fonctionnalité d’export, j’ai pu me familiariser avec les problématiques de la programmation d’un grand projet, ou plusieurs acteurs différent commit du code. Ce que l’on ne rencontre par forcément dans un projet en milieu scolaire. J’ai aussi du crée de la documentation pour le pipeline que j’ai créée, pour qu’elle puisse être reprise plus tard. J’ai eu la satisfaction de terminer les tâches que l’on m’a conféré et j’ai l’impression d’avoir énormément appris au cours de ces deux mois, non seulement d’un point de vue technique, mais aussi sur le fonctionnement en entreprise.

1. Plugin Blu Age Cobol

Afin de permettre à un développeur COBOL de compiler leur application dans un autre langage, Blu Age met disposition de leurs client un Plugin VSC. Ce plugin ajoute à l’IDE plusieurs commandes pour compiler leurs codes, par l’intermédiaire d’une requête http.

Plusieurs types d’opérations sont proposées par l’extension : D’une part, la compilation du code Cobol vers java, et d’autre part, la compilation de fichiers BMS uniquement, afin de générer une application web. Le fichier jar produit, issue de la compilation par l’extension, pourra ensuite être exécuter sur une fonction Lambda d’Amazone ayant une Layer contenant le Framework Velocity (c’est la layer dont nous avons construit la chaine de déploiement). (cf. contexte technique).

Lorsque le client appelle une de ces commandes sur Visual, une phase de vérification, va au préalable empêcher la compilation si le fichier sélectionné n’est pas du cobol, et ce, peut-importe la nature de la commande invoquée. Or, pour la compilation d’un fichier BMS, la présence d’un fichier Cobol n’est pas nécessaire.

L’objectif de cette tâche est de séparé les phases de vérification pour qu’elle soit adapter au type de compilation invoqué. La modification un fois effectué, j’ai généré à partir du code un fichier ‘vsix’ ce fichier pourra permettre à un utilisateur d’ajouté l’extension.

1. Plugin Blu Age Cobol

Les application CICS  (Customer Information Control System) sont des systèmes qui permet d'effectuer des [opérations transactionnelles](https://fr.wikipedia.org/wiki/Transaction_informatique) (en général consultation ou mise à jour de [bases de données](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es) ou de [fichiers](https://fr.wikipedia.org/wiki/Fichier_informatique)). Généralement codé en COBOL, c’est couramment ce type d’application qui doit être transcrite par l’extension de Blu Age. En plus des fichiers COBOL, des fichiers bms, pour l’apparence de l’application et un fichier textuel de description de ressource sont généralement présent. Lors de la compilation, ces fichiers doivent également être traité. Entre autres, le fichier textuel de description de ressource est parsé pour en une suite de requêtes SQL qui permet de former une base de données lié à l’application.

Au moment de la compilation, cette opération n’est pas visible pour l’utilisateur de l’extension. Et il n’a pas accès à la suite de requête SQL qui permet de crée la base de données.

L’objectif de cette tâche est d’ajouter une commande à l’extension Blu Age Cobol afin de récupérer un fichier SQL généré à partir d’un fichier textuel de description de ressources.

Tous d’abord, coté frontend, une commande nommée « Blu Age parsing CSD to SQL » a été créé en type script avec l’IDE Visual Studio Code. Lorsque celle-ci est invoquée, le fichier CSD sélectionné va être transmis au server COBOL par le biais d’une requête http.

Puis côté server, la requête est traitée : A l’aide des outils déjà présents, le fichier CSD réceptionné est traduit en SQL, zippé, puis renvoyé au client. (Développement en java, avec Eclipse).

Ensuite, à nouveau côté client, la réception des réponses du server a été adapter pour que le fichier soit récupéré correctement.

Le développement des mécanismes présenter a évidemment été inspiré des commandes de compilations Blu Age déjà présente. Mon travail à été d’ajouter à une architecture déjà présente, une fonctionnalité supplémentaire.

Le choix de cette solution est motivé d'un part, par sa flexibilité, mais également pour son cout très faible comparé au autre solution proposé. Enfin, l'atout de cette méthode comparativement l’utilisation de Jenkins par exemple est que l'on utilise quand même des outils Amazone, et les opération sont efectuer de manière interne au compte amazone de l'entreprise. En d’autres termes, on élimine la necessité de devoir se logger et donc de transmetre des identifiants ni en claire. Cette solutions propose le meilleurs compromis entre sécuité, fleibilité et cout.

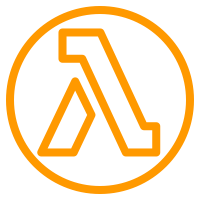
**Aperçu global du prototype**

Avant toute chose, . C'est la chaîne d'intégration. Cette chaine va compilé et tester le code. A l'issue de cette chaine, la nouvelle verion du framework va être chargé dans le Bucket Source (cf fig). Ce fichier, c'est ceui indiquer sur le shema sous le nom de "Layer File". C'est la que la chîne de de-ployment commence .L'action de déposé un fichier dans le bucket va alors déclancher le script de la Pipeline sur une Lambda. Cette fonction que l'on nome "MainPipeline", va récupéré le fichier, publié la Layer, la tester, et ajouter les permissions on compte AWS qui doivent avoir access. Toute ces opération sont effectuer dans la region par default (cannoniquement, en Virgini du Nord us-east-1). Puis avant de ce terminer, le scipt va déclancher de manire asyncrone (C'est à dire simultanément, comme des threads) N fonction lambda "Deployer". Chacune avec comme argument une regions différente. Leurs rôle, effectué le déployment de la layer dans la region qui leurs à été attitré. Elle vont donc à pari du bucket source, crée une copie de se bucket dans un bucket qui sera loalisé dans leurs régions, pour pouvoir publié le framework sous from d'une Layer dans leurs régions.

**Optimisation des performances :**

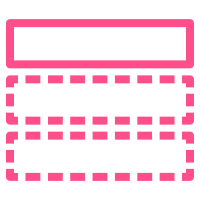
La chaîne présenter à la partie précédente démarre en série (de manière syncrone) une d'autres Lambdas effecuant la fonction "Depoyer". La fonction principal de la pipeline "ServerlessPipeline" va attendre qu'un deployer s'achève, soit qu'il récpère à réponse, pour démaré le deployer suivant. Hors, aws facture l'utilisation de la lambda selon sont temps d'utilisation. Il semble alors judicieux de réflechir à comment minimiser les temps d'attente. Pour évalué les performance de cette méthode d'organisation de l'execution des lambda, nous allons observé le temps moyens d'execution de la lambda en fonction du nombre de région dans laquelle on va la déployer. On cherchera à réduire au maximum cette durée afin de rendre notre solution plus economique.

**Schéma fonctionnelle de l’utilisation « serverless » du Framework Velocity**



***Fonction Lambda***

(environnement d’exécution de l’application)



***.jar***

(Issue de la compilation par l’extension Visual Studio code)

***Layer Velocity***

(dépendances)

***COBOL***

**Schéma fonctionnelle de la compilation par l’extension Blu Age Cobol**

Client utilisant Visual avec l’extension

Server de Blu Age

Attachement : project.jar

Réponse https

Attachement : project.zip

***Projet COBOL***

***Projcet.jar***

Attente d’une réponse

Requête https POST

Compilation

***Projcet.jar***

***Copie du Projet COBOL***