

Dossier de validation

U'DEV 2

Concepteur développeur d'applications numériques

Nom Prénom	LAERA Jérémie
Nom(s) Prénom(s) du ou des tuteurs	Emmanuel Lafitte
Acronyme de la certification IPI visée	CDAN
Niveau visé	6
Date de la soutenance	27/11/19
Lieu de la soutenance	EPSI/WIS BORDEAUX – 114 Rue Lucien Faure – 33000 BORDEAUX

Table des matières

1. Introduction.....	8
2. CGI et La Banque Postale : mes environnements d’alternance	9
2.1 CGI – Une multinationale reconnue	9
2.2 CGI – Bordeaux	10
2.3 Compréhension du métier visé au sein de CGI – Le développement de logiciels.....	11
2.3.1 Le métier de concepteur/développeur	11
2.3.2 Le métier d’analyste	11
2.3.3 Le métier d’Intégrateur	11
2.3.4 Le métier de Testeur	11
2.3.5 Communication	12
3. SIROCCO – Le référentiel Client et Contrat LBP	13
3.1 Problématique et Objectifs	13
3.2 L’équipe	13
4. Le Mainframe et Z/Os.....	15
4.1 Les types de programme mainframe	15
4.2 Les fichiers dans le mainframe	15
4.3 Connexion au Mainframe.....	16
4.3.1 TSO.....	16
4.3.2 UDC.....	17
4.4 COBOL.....	17
4.4.1 Histoire et Versions de COBOL	18
4.4.2 Environnement de développement pour le COBOL.....	18
4.4.3 Structure d’un programme en COBOL	18
4.5 PacBase.....	21
4.6 PTF	21
4.6.1 Définition	21
4.6.2 Exemple d’un programme PTF	21
4.7 Les Base de données sous le z/OS : DB2 & ARMIDE.....	23
4.7.1 DB2 & COBOL.....	23
4.7.2 ARMIDE : Un Base de Données de paramètres	23
4.8 JCL – Scripts et exécution de composants batch.....	23
4.8.1 Structure d’un JCL.....	24
4.9 Organigramme des traitements sous SIROCCO	26
4.10 Tester un composant.....	28

5. Compétences acquises au cours de l'année an Alternance U'DEV2	31
5.1 Compétences acquises au sein de CGI Le Haillan.....	31
5.1.1 Analyse, Conception et développement de programmes et composants mainframe	31
5.1.2 Evoluer un programme existant : modifier un algorithme sans générer de dysfonctionnement	31
5.1.3 Débogage de programme mainframe	32
5.1.4 Rédaction de tests unitaires et tests d'assemblage	32
5.1.4 Mise à disposition de composant(s) ou programme(s) vers les environnements supérieurs – Déploiement	33
5.1.5 Estimer mon Reste à faire (RAF) et contrôler les délais	33
5.1.6 Communiquer avec mon équipe et mettre à jour mes analystes sur mes avancées ou blocages.....	33
5.1.7 Concevoir des packages contenant des solutions logicielles automatisées	33
5.1.8 Livrer un logiciel conforme aux besoins du client	33
5.1.9 Respecter des contraintes et conventions de qualité de code en termes d'architecture logicielle.....	34
5.1.10 Clôturer une mission	34
5.2 Compétences acquises à l'extérieur de CGI (EPSI, Projets Personnels)	34
5.2.1 Conception de logiciels bénéficiant d'une architecture N'Tiers	35
5.2.2 Développement de programmes utilisant la Programmation Orientée Objet (OOP) – Exemple d'un blog pour Entreprises Sociales	35
5.2.3 Garantir un accès sécurisé aux données – Exemple illustré avec le blog pour Entreprise Sociale et Flask	41
5.2.4 Produire du logiciel en équipe.....	43
5.2.5 Concevoir des éléments logiciels opérationnels, génériques et réutilisables/partageables	43
5.2.6 Compétences transverses	47
6. Conclusion	49
7. Références	50
8. Annexes	51

Syllabus

Ce document a pour objectif d'expliquer et d'analyser la progression réalisée par alternance dans le cadre de la formation U'DEV 2 en France. J'ai été amené à effectuer une alternance allant du 03 décembre 2018 au 29 novembre 2019 au sein de l'entreprise CGI au située au Haillan près de Bordeaux. L'intitulé exact de mon poste est développeur, concepteur et débutant en analyse Mainframe pour un système d'information bancaire. Au cours de cette année, un cursus a également été suivi à l'école afin de nous apporter des notions complémentaires en conception et développement de solutions informatiques.

Le plan suivi dans ce document me permettra de mettre en avant les atouts et compétences acquis ou en cours d'acquisition au cours de l'année écoulée. Le fil directeur de cet écrit est un tableau de compétences disponible en annexe. Je m'efforcerai donc au travers des différentes parties du document d'explicitier les compétences que j'ai acquises en lien avec ce tableau de compétences.

Le plan suit la logique suivante :

- 1) Introduction
- 2) CGI et La Banque Postale
- 3) SIROCCO et mon équipe
- 4) La technologie Mainframe
- 5) Enumération des Compétences Acquisées durant l'année U'DEV

Ce document a donc pour objectif principal de mettre en avant les qualifications et compétences requises pour être un développeur/analyste débutant. Il fonctionnera comme un justificatif détaillé ciblant 5 blocs de compétences exigés par la formation qui sont :

- 1) Qualité et sécurisation du code utilisé
- 2) Audit, conception, méthode de projet
- 3) Réalisation d'applications logicielles
- 4) Communiquer avec les acteurs du projet
- 5) Adapter l'environnement d'exécution, échanges des données avec d'autres logiciels

Cette nomenclature est officiellement fournie par l'école IPI.

Remerciement

Je tiens à remercier :

- Emmanuel Lafitte, mon tuteur et maître de stage d'alternance.
- Julien Réau et Patrice Eyraud, mes chefs de projet qui ont su m'apporter les connaissances managériales et fonctionnelles concernant les outils administratifs du projet.
- L'équipe support de notre plateau et plus particulièrement Raymond Gibault, qui a su prendre du temps afin de m'éclaircir sur certaines notions techniques liées à la technologie mainframe.
- L'équipe support et experte, Raymond Gibault, Vincent Giral, Sébastien Fourcade et Arthur Caledec pour les formations CGI qu'ils ont fournies.
- L'équipe avec laquelle j'ai travaillé cette année, qui a pu m'intégrer au fur et à mesure au sein du projet, et partager son savoir et son expérience.
- Et de manière plus globale, tout le plateau SIROCCO et Grand-Ouest, au sein duquel j'ai réalisé mon alternance. Ils m'ont permis d'avoir un cadre de travail très agréable, une ambiance sérieuse et conviviale, propice à une bonne productivité et à une évolution certaine pour ma part.
- Rodolphe, Julien et Emmanuel du projet, pour la relecture sérieuse de cet écrit.
- François, pour son accompagnement lors de la rédaction de cet écrit.

Sigles & Abréviations

IPI = Institut de Poly-Informatique

CGI = Conseillers en Gestion et Informatique

CdS = Centre de Services

LBP = La Banque Postale

DCG = Dossier de Conception Générale

DCOD = Dossier de Conception Détaillée

COBOL = Common Business Oriented Language

JCL = Job Control Language

SQL = Structured Query Language

CICS/TP = Customer Information Control System

SA = Service Applicatif

OOP = Object-Oriented Programming

IDE = Integrated Development Environment

Partie 1

1. Introduction

Au cours du cursus U'DEV2 allant de septembre 2018 à fin novembre 2019, j'ai été amené à effectuer une période d'alternance allant du 03 décembre 2018 au 29 novembre 2019 au sein de l'entreprise Conseillers en Gestion et Informatique (CGI). Cette alternance fait partie intégrante d'un contrat acté par la filière U'DEV avec l'école EPSI, Pôle Emploi et CGI. Nous sommes une trentaine à représenter la promotion U'DEV2 à Bordeaux, et nous avons tous eu à effectuer un contrat d'alternance visant à éveiller, développer et valider des compétences professionnelles, fonctionnelles et techniques, dans le domaine du développement de logiciel en Sciences de l'informatique. Pour ma part, ce contrat m'a permis de participer à la réalisation de logiciels et composants « mainframe » pour un système d'information bancaire (SI).

En premier lieu, pour introduire ce dossier de validation et rapport d'alternance, je présenterai l'entreprise CGI en quelques lignes, et plus précisément CGI Bordeaux et notre client La Banque Postale, ainsi que l'équipe au sein de laquelle j'ai effectué le stage d'alternance de bout en bout.

En second lieu, et en lien avec la première partie, je m'attellerai à décrire le projet pour lequel j'ai travaillé. Pour cela, je parlerai des problématiques qui ont mené à la naissance de ce projet. Je mettrai en évidence les différents aspects et difficultés liés à la gestion de ce projet.

Ensuite, je détaillerai les différentes technologies utilisées par le projet pour expliquer la formation et montée en compétence que j'ai dû concrétiser afin de résoudre les challenges professionnels rencontrés au quotidien à CGI.

En corrélation avec cette montée en compétences, j'exposerai le travail que j'ai été amené à réaliser au cours de l'année en expliquant les contraintes techniques et fonctionnelles qui m'ont été imposées. Je procéderai également à une mise en évidence des atouts que ces tâches m'ont permis de développer, ainsi que les évolutions possibles. J'accentuerai finalement le caractère sérieux de mon travail afin de démontrer qu'être développeur signifie aussi savoir s'autoévaluer et avoir un sens critique développé.

Enfin, je prendrai du recul par rapport au stage réalisé en entreprise, afin de m'attarder sur les compétences acquises avec l'école EPSI et au cours de projets personnels, tout en expliquant leur intérêt personnel et professionnel. Cette partie a pour objectif de démontrer les compétences acquises hors entreprise liées aux composantes de la certification fournie par l'IPI.

Je conclurai en mettant en avant ce que l'alternance ainsi que l'année U'DEV2 m'ont apporté au niveau professionnel et personnel.

Partie 2

2. CGI et La Banque Postale : mes environnements d'alternance

Mon alternance U'DEV2 s'est déroulée au sein de CGI situé au Haillan, en métropole Bordelaise. Dans les sous-parties à venir, je vais décrire l'entreprise CGI, ainsi que le client pour lequel j'ai travaillé. Ces explications ne sont pas exhaustives mais ont pour but de donner une définition simple et concise de l'environnement à l'intérieur duquel j'ai évolué.

Un des plus importants clients et partenaire de CGI Bordeaux est La Banque Postale (abrégée LBP). Il s'agit du client pour lequel mon projet se dédie. CGI est prestataire de la LBP et cette relation date du temps de la société Logica.

2.1 CGI – Une multinationale reconnue

CGI, ou CGI incorporation (CGI Inc.), est une société de services en informatique (SSI) d'origine canadienne. On peut également retrouver le terme d'ESN signifiant Entreprise de Services du Numérique. La société fut fondée en 1976 par Serge Godin et André Imbeau, à Québec. Au tout début, CGI pouvait signifier « *Conseillers en Gestion et Informatique* » ce qui peut se traduire en anglais par « *Consultants in Management and information technology* ». L'entreprise a par ailleurs su hériter de certains de ces termes puisque sur nos outils administratifs nous pouvons apercevoir les statuts de nos collègues analystes comme étant « *consultant* » ce qui reflète bien l'héritage de la philosophie d'origine de la société Canadienne.

Il fallut 10 ans à CGI avant de monter sur la scène internationale. Un fait notable est leur introduction en bourse à Toronto en 1986 (abrégée TSX en anglais). En 1998, CGI doubla pratiquement en terme d'effectif avec l'acquisition de Bell Sygma. En 2001, l'entreprise racheta IMRGlobal pour la somme de 438 millions de dollars ce qui la propulsa davantage sur l'échelle internationale. Suite au rachat de la société Logica en 2012, les effectifs de CGI passent de 30 000 à 70 000. En France Logica est dès lors englobée par CGI ainsi que ces effectifs.

L'ESN canadienne a pour objectif principal de proposer à ses clients, provenant de secteurs d'activités diversifiés, toute une panoplie de services liés au domaine du numérique (conseil et gestion de projet, conception et réalisation d'outils informatique, maintenance ou encore formation spécialisées).

En 2018, CGI lance l'école du développeur U'DEV afin de répondre à une forte demande de postes de développeur/concepteur; école à laquelle j'appartiens et qui fête sa deuxième année.

Dans un futur proche, CGI ambitionne d'étendre son périmètre en Europe, et de « soutenir son développement en France » en recrutant environ 3000 ressources en 2019(<https://www.cgi.fr/fr-fr/qui-sommes-nous/cgi-france>).



Actuellement, CGI fait partie des 100 plus grandes SSI au monde. De manière plus spécifique, CGI se classe en 20^{ème} position dans la liste des 100 plus importantes marques canadiennes (2018). CGI se classe également 15^{ème} dans la liste des 100 entreprises qui investissent le plus en recherche et développement (2018).

Son évolution peut se remarquer, comme l'indique son logo :



Figure 1 - Les différents Logo de CGI en affichage évolutif

2.2 CGI – Bordeaux

Avant d'être ce qu'elle est CGI a auparavant racheté l'ancienne entreprise Européenne de services en informatique Logica. En France, Logica comprenait environ 9200 collaborateurs (Wikipédia). Suite à ce rachat, les effectifs CGI en France et à plus grande échelle en Europe grimpe une nouvelle fois.

L'entreprise CGI s'est installée à Bordeaux au début de l'année 2011 et emploie environ 1000 membres répartis sur 3 Business Unit. Au sein de cette entreprise, ces 3 Business Units (BU) sont : Grand-Ouest (GO), France Global Delivery Center (FGDC) et Transport Public Service Human Resource (TPSHR).

- Grand Ouest prend en charge les projets de tous les secteurs d'activités situés dans le périmètre de l'Ouest et le Sud-Ouest.
- FGDC s'occupe des projets de tous les secteurs d'activités à l'échelle nationale comparé à Grand-Ouest.
- TPSHR gère tous les projets liés aux transports, aux services publiques et aux ressources humaines.

Avant cette installation en 2011, la société Logica évoquée plus haut était présente, et était le prestataire d'origine de La Banque Postale (LBP).

L'équipe que j'ai intégrée appartient à la Business Unit (BU) de Grand-Ouest. Mon travail en son sein se rapporte au domaine du développement de logiciels et composants informatique ainsi que de leur conception et intégration dans nos environnements.

Le travail que j'effectue se focalise sur le Centre De Services Référentiel (CDS REF) et en particulier sur la technologie de mainframe pour le compte de son client La Banque Postale. Le CDS REF a pour objet de travail principal les référentiels et notamment Sirocco, un des principaux référentiels Clients et Contrats de LBP. C'est le projet que j'ai intégré.

Dans la suite du document, j'emploierai par conséquent le terme de SIROCCO pour référencer le nom de mon projet.

2.3 Compréhension du métier visé au sein de CGI – Le développement de logiciels

Le métier visé est celui de concepteur/développeur, intégrateur et testeur au sein de CGI. Ce métier consiste à mettre en œuvre une approche des métiers adjacents figurant dans les sous-parties de cette section dédiée à la compréhension du métier ciblé.

2.3.1 Le métier de concepteur/développeur

Un concepteur ou développeur de solutions informatiques est un expert des langages informatiques. Il doit être capable de traduire toute demande émanant du client en « lignes de code ». Les demandes clients sont généralement émises et rédigées sous forme de « cahier des charges ». Ces lignes de code constituent le programme ou logiciel. De nos jours, un développeur fait partie des travailleurs les plus recherchés en raison de la forte demande provenant des secteurs du numérique. Il doit posséder la capacité d'adaptation, car les technologies et langages peuvent varier selon les systèmes et entreprises. Ils peuvent également évoluer. Le développeur doit par conséquent savoir étendre ses compétences.

Un développeur peut être amené à participer en amont à l'analyse spécifique des besoins du client et des utilisateurs du produit à développer. Cette action rentre dans le cadre d'un métier qualifié d'analyste. J'expliquerai ma compréhension de ce métier dans le paragraphe à venir.

2.3.2 Le métier d'analyste

Outre le fait de réaliser le code après avoir perçu les spécificités techniques d'un besoin, un développeur peut également endosser le rôle d'analyste.

Un analyste est avant tout un programmeur. Cependant son axe de travail va davantage se focaliser sur la traduction du besoin client en un algorithme. Cet algorithme sera traduit en lignes de codes par le développeur.

La notion d'analyste est précise, car en général pour être un bon analyste, de l'expérience en tant que développeur est requise. Il existe actuellement sur mon plateau des analystes développeurs (qui endossent à la fois l'analyse de besoins client et l'écriture de code) ainsi que des développeurs (personne qui réalise le code). À l'heure actuelle, je me trouve plus dans le statut de développeur que celui d'analyste; cependant je commence à gagner des compétences en analyse mainframe depuis quelques mois.

2.3.3 Le métier d'Intégrateur

Une autre fonction que je suis amené à effleurer est celle d'intégrateur. En informatique, l'intégrateur est en réalité un « voisin » du développeur. Son rôle va essentiellement tourner autour de l'assemblage des différents composants d'un système. Des installations peuvent s'effectuer sur le site du client par l'intégrateur.

2.3.4 Le métier de Testeur

Il s'agit de réaliser les phases de tests unitaires (TU) : ces tests consistent à tester les instructions ou blocks de code d'un programme dans ses moindres détails. L'environnement mainframe de LBP implique jusqu'à ce jour une manière singulière d'effectuer les TU car ceux-ci ne sont pas

automatisés par des classes ou outils. Dans ce dossier je m'efforcerai donc d'expliquer en quoi consistent cette singularité. Il en découlera qu'effectuer des TU en mainframe, est relativement différent de la réalisation de TU dans le cadre d'une application web ou Mobile.

2.3.5 Communication

Le compte-rendu de toutes les actions précédemment évoquées à l'analyste ou expert en relation avec la demande à traiter est un facteur primordial du travail d'un développeur. Il est important de faire des points réguliers avec son analyste et/ou son responsable application.

Selon moi, être un concepteur/développeur d'applications logicielles, c'est avant tout savoir prendre du recul et comprendre le besoin exprimé par le client. Un développeur se doit d'être en phase avec la demande et de savoir déterminer la logique que le composant à produire impliquera, que ce soit sur le traitement même du programme ou bien sur les chaînes de traitements dans lequel il se trouvera.

Dans le cadre de ce dossier de validation, je vais dans un premier temps décrire fonctionnellement le projet auquel j'ai été affecté. En effet, il est essentiel d'arriver à cerner les différentes fonctionnalités principales de ce dernier afin d'être en capacité de rentrer dans les détails techniques. La section suivante s'articulera donc autour de la description fonctionnelle du référentiel SIROCCO.

Partie 3

3. SIROCCO – Le référentiel Client et Contrat LBP

SIROCCO signifie **S**ystème **I**nformation **R**éférentiel **O**rienté **C**lient **C**ontrat.

L'objectif de cette section est fournir un résumé concis à propos du référentiel SIROCCO.

3.1 Problématique et Objectifs

Le référentiel SIROCCO fut créé afin de constituer un référentiel Clients/Contrats centralisé des services financiers de la Banque Postale. Cette ambition est due au fait qu'auparavant chaque domaine de production (Dépôt, Epargne, Cartes...) était géré de manière autonome. De plus, chacun d'entre eux possédait son propre référentiel Clients/Contrats, avec ses fonctionnalités internes : le problème fut de centraliser les informations au sein d'une et une seule application. L'idée de rassembler ses fonctionnalités avait pour but de créer un grand système de bases de données (BDD), englobant les données des clients et contrats pour la banque.

SIROCCO fixe par conséquent plusieurs objectifs tels que :

- L'offre d'un nouveau noyau de fonctionnalités communes pour répondre aux objectifs commerciaux et organisationnels.
- La provision d'une base d'informations Clients/Contrats commune à tous les domaines.
- Améliorer et personnaliser la relation avec le client.
- Standardiser la gestion des clients.
- Maintenir la cohérence des données.
- Faciliter la souplesse organisationnelle.

SIROCCO est capable de gérer les entités principales suivantes en terme de base de données : Les Clients et Contrats, ainsi que les informations qui leur sont relatives. De cette manière, l'application offre la possibilité de visualiser les informations sur un client ou un contrat de manière globale et centralisée.

De manière générale, le CDS REF regroupe un éventail d'applications, ayant chacune un rôle et des fonctionnalités propres. J'ai effectué la totalité de mon alternance au sein du projet SIROCCO, cependant il m'est arrivé de communiquer avec d'autres applications lors de développements de modules COBOL.

3.2 L'équipe

Contrairement à certains plateaux, le plateau CDS englobe plusieurs projets en relation avec SIROCCO. Cette application prend en charge les conceptions et réalisations de modules COBOL, mais aussi les traitements et analyses d'incidents liés à des anomalies rencontrées.

L'équipe au sein de laquelle j'ai évolué durant ma période d'alternance comprend :

- Un directeur de projet.
- Deux chefs de projets et managers.
- Un expert fonctionnel.
- Un expert technique.
- Une équipe dédiée aux SA¹, composé d'une responsable d'application ainsi que d'un analyste.
- Une équipe MCO² composée d'analystes/développeurs et testeurs.
- Des responsables Applications (RA).
- Plusieurs analystes mainframe.
- Quatre développeurs mainframe dont je fais partie.
- Une équipe délocalisée à Nantes composée d'une hiérarchie proche de la nôtre.
- Une équipe au Canada.

Au cours de cette période, j'ai eu l'opportunité de travailler avec la plupart des collègues mentionnés ci-dessus.

Ceci conclut la description fonctionnelle du référentiel SIROCCO sur lequel j'ai travaillé au cours de mon stage. Dans la partie suivante, je vais m'efforcer de détailler les technologies que j'ai appris à utiliser au cours de cette période d'alternance.

Ce sont des technologies relativement anciennes en terme d'informatique. Néanmoins, dû à des fortes demandes, causées notamment par les départs de fin de carrière et contrats neufs avec les clients, ces technologies sont redevenues prisées sur le marché du travail. Elles répondaient et répondent toujours à un besoin particulier à savoir la gestion informatique des systèmes bancaires et assurances. C'est pour cette raison que de nouvelles formations voient le jour afin de former de nouveaux informaticiens dans ces domaines. Un exemple est le centre de formation Ib localisé à Paris qui offre des formations payantes IBM z/OS.

En effet, grâce à leur efficacité et robustesse, les mainframes z/Os ont été et demeurent performants.

¹ Service Applicatif

² MCO ou Maintenance en conditions opérationnelles : représente une équipe dédiée à gérer les bugs ou anomalies rencontrés, notamment en production.

Partie 4

4. Le Mainframe et Z/Os

Les fichiers et programmes sont organisés d'une manière singulière. Nous devons respecter un positionnement lors du placement de nos données ou de code. Ce positionnement est géré par un colongage. J'expliquerai comment ce dernier transparaît en COBOL et dans l'écriture des JCL.

4.1 Les types de programme mainframe

J'ai interagi avec deux types de programmes :

- 1) Les programmes Batch. Ce sont des programmes exécutant des traitements séquentiels sur des fichiers. Ils sont répandus et permettent de faire transiter des données sous forme de fichiers ou table de base de données d'application en application.
- 2) Les CICS. Chez SIROCCO ce sont des Services Applicatifs ou SA. Ils ont pour but de créer des interactions homme/machine avec des traitements d'écrans en temps réel. Les Services Applicatifs sont divisés en plusieurs couches logicielles afin que chaque composant possède un rôle (accès aux base de données, traitement de données, envoie de résultat de traitement vers le composant responsable de l'affichage...). Interagir avec un écran et avoir une réponse en mainframe se nomme « conversation ». Une conversation a lieu entre un homme et la machine.

4.2 Les fichiers dans le mainframe

Avant d'entrer dans le vif du sujet, il est important d'expliquer comment fonctionnent le système de fichiers sous le mainframe. Nos fichiers se trouvent dans un système virtuel intitulé système MVS. Afin de rester le plus clair possible je me focaliserai sur les fichiers les plus couramment rencontrés.

Il existe différents types de fichiers sous le mainframe :

- Les fichiers VSAM (Virtual Storage Access Method) sont des fichiers utilisant des index et données. Ces fichiers sont généralement très utilisés pour définir les tables de base de données DB2.
- Les fichiers séquentiels sont des fichiers contenant des enregistrements tout comme un fichier texte sous Windows. Ils peuvent être de taille Fixe Bloqué noté FB ou Variable Bloqué noté VB.
 - Fixe bloqué signifie que chaque enregistrement possède une taille maximale fixe. Cette taille est exprimée en octets.
 - Variable bloquée signifie que le fichier possède une taille maximale définie mais que si les enregistrements sont plus courts, alors les enregistrements en question seront

arrêtés au niveau de la dernière alimentation faite afin de supprimer tous les espaces inutiles en fin de ligne.

- Les fichiers à génération sont des fichiers qui vont se générer automatiquement en se basant sur un modèle établi appelé enveloppe. Elle contient les caractéristiques du fichier (taille, format...). C'est pour cette raison que l'on entend également parler de fichier « enveloppe ».
- Les fichiers partitionnés (PDS en anglais pour Partitionned Data Set) sont l'équivalence des dossiers ou répertoires. Ils sont définis selon certaines caractéristiques afin de contenir d'autres fichiers. Ces derniers sont appelés des membres.

Cette brève description n'est pas exhaustive car le système de fichiers et la notion de MVS sont en réalité beaucoup plus complexes et proviennent du fait que le mainframe souhaitait à l'origine faire travailler de la mémoire virtuelle afin d'effectuer plusieurs travaux de manière simultanée. Pour toute personne envieuse d'en savoir plus, la documentation officielle d'IBM fournit un détail sur les fichiers dans le mainframe.

4.3 Connexion au Mainframe

Afin de nous connecter au mainframe, il nous faut d'une part une interface (émulée ou non), puis d'une autre part un branchement sur un processeur mainframe. J'utilise deux solutions que j'expliquerai brièvement : TSO et UDC.

4.3.1 TSO

TSO signifie Time-Sharing Option et est un interpréteur de terminal qui est branché avec le système MVS d'IBM. Après certaines évolutions, il est aujourd'hui appelé TSO/E pour Time-Sharing Option Extensions. Time-Sharing signifie que plusieurs utilisateurs peuvent accéder au système d'exploitation de manière simultanée. TSO est très utile et rentable car les commandes sont très rapides à s'exécuter et il englobe des utilitaires IBM permettant de faire tout type de manipulation sur des composants ou fichiers. Il englobe également les utilitaires DB2 Platinum et Spufi ainsi que l'utilitaire de débogage Xpediter.

L'interface d'un terminal TSO se présente dans la manière suivante :

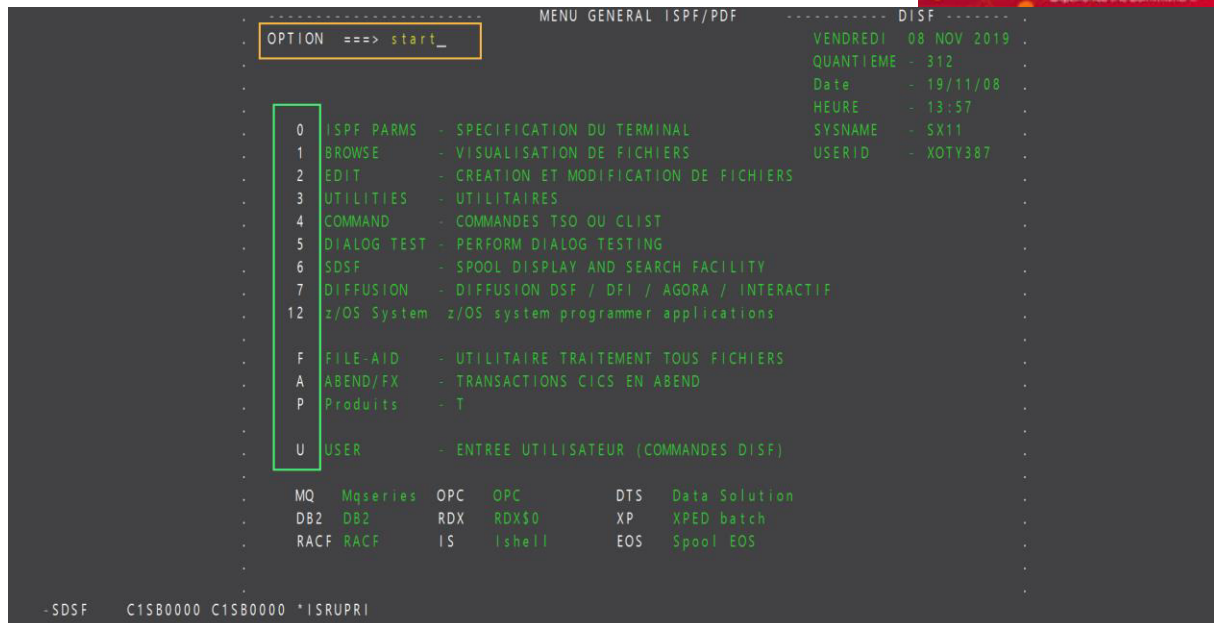


Figure 2- Ecran d'accueil TSO

La commande **start** lance une nouvelle fenêtre comme celle présente sur la figure 2. Les différentes options de démarrage sont encadrées en vert.

TSO permet de faire ce qu'un IDE³ moderne permet de faire en C++ ou Java. Toutes les fonctions nécessaires à du développement, test et débogage sont installées.

4.3.2 UDC

UDC signifie Usine de Développement COBOL. Il s'agit d'un nouvel outil mis en place afin de remplacer progressivement la technologie PacBase mentionnée plus bas, qui n'est à ce jour plus maintenue par IBM.

UDC est en réalité une surcouche de l'IDE Eclipse. Il possède donc des propriétés graphiques très similaires à ce dernier. Néanmoins ce n'est pas son seul atout. Un autre objectif d'UDC est d'offrir des fonctionnalités égales à celles de TSO. J'ai donc la possibilité d'utiliser soit TSO, soit UDC en fonction de mes habitudes. Une unique différence réside dans l'obligation de développer les programmes COBOL sous UDC et non pas sur TSO.

J'ai cependant plus souvent recours à TSO pour les manipulations de fichiers et tests car les commandes sont rapides et l'affichage est pertinent et instantané.

4.4 COBOL

COBOL signifie Common Business Oriented Language. Le langage a fait ses débuts en 1959 par le biais du CODASYL Committee (Conférence sur les Data Systems Language).

³ Integrated Development Environment

Ce langage est spécifiquement utilisé pour ce que l'on nomme Business Applications qui sont des logiciels et applications dédiés aux entreprises liées aux industries financières et bancaires. En effet, COBOL est capable de manipuler de larges quantités de données en parallèle.

ANSI (American National Standard Institute) a travaillé pour améliorer le langage COBOL, et de nombreuses évolutions ont vu le jour en 1965 et 1974.

En 1985, une version améliorée paraît avec de nouvelles fonctionnalités. C'est en 2002 que la version actuelle paraît avec l'apparition du paradigme de langage orienté objet. Les objets arrivent en COBOL mais sont alors peu utilisés au sein des Entreprises et Sociétés de Services (SSI).

Sous SIROCCO, je n'ai pratiquement pas eu à créer ou modifier de composant en COBOL natif. Ceci est causé par l'arrivée du Framework PTF.

4.4.1 Histoire et Versions de COBOL

Pour rappel, COBOL apparut sur le marché en 1959 et fut développé par CODASYL. En 1960, les compilateurs COBOL deviennent disponibles. ANSI travailla durement à améliorer le COBOL dans les années 60 et 70 ce qui mena à la version de 1974 : cette version est référée en tant que COBOL 74. En effet, les versions de COBOL caractérisent l'appellation donnée au langage, par exemple l'évolution de 1985 encore beaucoup utilisée en entreprise et dénommée **COBOL 85**.

Le COBOL est capable de développer des applications portées par les consistances et assistance en connectivité fourni par les systèmes IBM. C'est le cas pour COBOL 370, COBOL for MVS et VM, COBOL for OS/390. Actuellement, la dernière version de COBOL est appelée **Enterprise COBOL**. Il est également possible de développer en COBOL sur une machine Unix ou Windows via un émulateur et terminal. Ce processus s'effectue en 2 étapes (étapes brièvement expliquées auparavant).

4.4.2 Environnement de développement pour le COBOL

Pour pouvoir coder en COBOL, il nous faut réaliser 2 actions :

- 1) Connecter notre système à un processor de type Mainframe.
- 2) Se connecter à une interface de type Mainframe.

C'est dans cette optique que TSO et UDC font leur apparition.

4.4.3 Structure d'un programme en COBOL

Tous les programmes COBOL respectent la même structure. Un programme COBOL se divise en 4 unités principales appelé Divisions en anglais.

4.4.3.1 Identification Division

La première division est l'identification division. Elle contient les informations identifiant le programme. Ceci inclut le nom du programme, le nom de l'auteur, la date d'écriture et de compilation du programme en plus du niveau de sécurité du programme.

4.4.3.2 Environment Division

La deuxième division se nomme Environment division. Elle contient les informations concernant l'environnement du programme. Elle se divise en 2 parties :

- 1) La Configuration Section qui décrit l'environnement dans lequel le programme est exploité. Cela peut être par exemple une utilisation de signe monétaire, un signe décimal particulier comme la virgule ou bien alors un jeu de caractères à spécifier.
- 2) L'Input-Output Section qui décrit les fichiers que le programme utilisera en explicitant le type de fichier (Fixe Bloqué ou Variable Bloqué) ainsi que le chemin et méthode d'accès.

4.4.3.3 Data Division

La troisième division est la Data Division. Comme son nom en anglais l'indique, cette division s'occupe d'instancier les variables que l'on utilise en mémoire. Elle se divise en 3 sections :

- 1) La File Section (FS) : la FS est utilisée afin de fournir la définition des fichiers utilisés par le programme.
- 2) La Working-Storage Section (WS) : la WS permet de définir toutes les variables intermédiaires ou temporelles du programme.
- 3) La Linkage Section : elle sert à faire appel à d'autres programmes depuis notre programme même.

Les formats de données en COBOL sont les suivants :

- a) Numérique : les valeurs peuvent contenir des chiffres et des signes (+ ou -) dans le cas où notre variable est un signé.
- b) Alphabétique : les valeurs contiennent des lettres. Les valeurs sont écrites entre guillemets simples.
- c) Alphanumérique : les valeurs peuvent contenir des chiffres et lettres. Les valeurs sont écrites entre guillemets simples.
- d) Constantes figuratives : ce sont des valeurs prédéfinies dans le langage COBOL. Certains exemples sont : ZERO (remplit une variable numérique par des 0), SPACE (remplit une variable alphanumérique par des espaces), LOW-VALUE (valeur hexadécimale la plus basse i.e. x'00') ...

Il faut utiliser le mot clé PICTURE ou PIC en abrégé dans l'intention de spécifier un format de données lorsque nous déclarons nos variables. Ce mot clé est suivi d'un spécificateur de format de données. Il y en a trois. « 9 » est utilisé pour déclarer une variable de type numérique, « A » est employé pour déclarer une variable de type alphabétique et « X » est utilisé pour déclarer une variable de type alphanumérique.

4.4.3.4 Procedure Division

La quatrième et dernière division est la procédure division. C'est ici que toute la logique ainsi que les algorithmes du programme se produisent. C'est ici que j'écris mes traitements (conditions, boucles, fonctions...). Elle contient donc toutes les instructions du programme.

Voici un exemple type de programme COBOL :

```

1 IDENTIFICATION DIVISION.
2 PROGRAM-ID. EXEMPLE.
3 ENVIRONMENT DIVISION.
4 DATA DIVISION.
5     WORKING-STORAGE SECTION.
6     * Variable are declared in this section
7     VAR1                                PIC X(8).
8     VAR2                                PIC X(10)
9                                     VALUE '10.11.2019'.
10    VARNUM                               PIC 9(8).
11    VARNUM2                             PIC 9(8)
12                                     VALUE 99994444.
13    VAR-MSG                             PIC X(60)
14                                     VALUE 'Résultat de la variable
15                                     'VAR1 après le traitement: '.
16    VAR-MSG2                             PIC X(60)
17                                     VALUE 'Résultat de la variable
18                                     'VARNUM après le traitement: '.
19 PROCEDURE DIVISION.
20 MAIN-PARA.
21     * Statements are carried out here
22     * This will grant VAR1 the value -> '20191110' (Date SSAAMMJJ)
23     MOVE VAR2(7:4)                      TO VAR1(1:4).
24     MOVE VAR2(4:2)                      TO VAR1(5:2).
25     MOVE VAR2(1:2)                      TO VAR1(7:2).
26     * This will grant VARNUM the value -> 44449999
27     MOVE VARNUM2(1:4)                   TO VARNUM(5:4).
28     MOVE VARNUM2(5:4)                   TO VARNUM(1:4).
29     * This will display each variable after the previous instructions
30     DISPLAY VAR-MSG.
31     DISPLAY VAR1.
32     DISPLAY VAR-MSG2.
33     DISPLAY VARNUM.
34
35 MAIN-PARA-FIN.
36 STOP RUN.

```

Figure 3 - Programme COBOL simple effectuant des formatages de données

Sur la figure 3, se trouve un programme COBOL de type exemple écrit sur VS Code à l'aide d'un plugin COBOL configurable sur l'éditeur. Ce programme applique un algorithme permettant de formater une date en plus d'une valeur numérique :

- Des lignes 1 à 4 se trouvent les 3 premières divisions.
- De la ligne 5 à la ligne 18, je déclare une WORKING-STORAGE SECTION suivie de la déclaration des variables que je vais manipuler dans ma PROCEDURE DIVISION.

- En ligne 20 puis ligne 35 se trouve une étiquette⁴. A l'intérieur de celle-ci, de la ligne 23 à la ligne 28 se trouvent les traitements du programme. J'utilise le mot réservé MOVE, qui me permet de déplacer le contenu d'une variable vers une autre.
- De la ligne 30 à la ligne 33, j'utilise le mot clé DISPLAY me permettant d'afficher les variables après les traitements effectués juste avant. Un DISPLAY peut en général être visualisé dans une console ou un terminal⁵ ou alors dans une carte SYSOUT paramétrée dans un JCL d'exécution.

4.5 PacBase

PacBase est un AGL⁶ (Générateur de code source COBOL), et a été utilisé pendant plusieurs années au sein de différents projets mainframe.

J'ai été amené à utiliser PacBase essentiellement pour effectuer des modifications de programmes. Aujourd'hui, nous utilisons un Framework de développement COBOL récent mis en place sur nos environnements de travail : PTF. PacBase n'est officiellement plus maintenu par IBM. Les programmes écrits en PacBase migrent progressivement en COBOL à cet effet.

4.6 PTF

4.6.1 Définition

PTF est un Framework de développement COBOL regroupant un éditeur : RDZ sous UDC qui est basé sur Eclipse, et un dictionnaire de données PTF. Il a été mis en œuvre récemment à CGI et chez son client LBP. Le SI LBP a fait l'objet d'une migration progressive de ses composants, par vagues successives. Suivant l'application, j'ai été amené à utiliser les deux technologies i.e. PTF et PACBASE.

L'outil de développement PTF a été pensé pour éviter d'avoir à écrire tout le code COBOL à la main. Des commentaires sur les programmes sont également générés en début de programme et sont encadrés par une balise PTF qui ne doit en aucun cas être altérée sous peine de déclencher des anomalies. Les informations telles que les fichiers en entrée et sortie ainsi que les macros utilisées apparaissent dans cette balise.

4.6.2 Exemple d'un programme PTF

La figure ci-dessous montre un exemple de début de programme PTF type généré après le renseignement de différents champs nécessaires à sa génération:

⁴ Ou Paragraphe.

⁵ C'est le même principe qu'un System.out.println() en Java ou console.log() en JavaScript.

⁶ AGL = Atelier de Génie Logiciel.


```

1  <cartouche> 02                                SKLT
2  IMPORTANT : afin d'assurer la maintenance du programme via
3  la fonctionnalité d'import, hormis la zone commentaires,
4  le contenu du cartouche ne doit en aucun cas être altéré.
5  ****
6  ***** CARACTERISTIQUES DU PROGRAMME *****
7  *****
8  Programme      : Nom du programme | Date de génération et heure
9  Libellé prog   : Libellé du programme
10 Cinématique    : Type du programme (Batch ou CICS)
11 DOM / APPLI    : Bibliothèque du programme
12 *****
13 ***** COMMENTAIRES DU PROGRAMME *****
14 *****
15 Auteur: Jérémie LAERA (identifiant user)
16 =====
17
18 Ce programme a pour fonction de montrer le fonctionnement du
19 Framework COBOL PTF.
20
21 *****
22 ***** ENTREES / SORTIES *****
23 *****
24 -----|----|-----|-----|---|-----|-----|-----
25 DDNAME  |I/O|COPY  |PREFIXE|FMT|STRUCT.|SUPPORT |MOD.LECT.
26 -----|----|-----|-----|---|-----|-----|-----
27 XXXXXX  |E  |XXXXXXX|XXXX   |V  |        |FI      |PA
28 XXXXXX  |E  |XXXXXXX|XXXX   |F  |        |FI      |PA
29 XXXXXX  |S  |XXXXXXX|XXXX   |F  |        |FI      |
30 XXXXXX  |S  |XXXXXXX|XXXX   |F  |        |FI      |
31 XXXXX   |S  |XXXXXXX|XXXX   |F  |        |FI      |
32 -----|----|-----|-----|---|-----|-----|-----
33 XXXX : Libellé du fichier 1
34 XXXX : Libellé du fichier 2
35 XXXX : Libellé du fichier 3
36 XXXX : Libellé du fichier 4
37 XXXX : Libellé du fichier 5
38
39 Aucune donnée en rupture et/ou synchronisation
40

```

Figure 4 - Structure type d'un squelette de programme PTF

Les informations générées par le Framework figurent donc à l'intérieur d'une balise PTF prénommée « cartouche ». Une petite note d'information se situe juste en dessous et précise qu'il est interdit de modifier manuellement la zone à l'intérieur de la cartouche, sauf en cas d'ajout de commentaire. Nous pouvons d'ailleurs le remarquer car sur la droite figure le tag ***FWK** (cadre rouge) qui indique l'origine de génération chaque ligne causée par le Framework.

Les **caractéristiques du programme** (cadre vert clair) figurent à la suite et détaillent le sous-environnement dans lequel ce programme doit fonctionner, le code application auquel il est rattaché, la bibliothèque à laquelle il est affiliée, son libellé et son type (batch ou CICS dans la plupart des cas).

4.7 Les Base de données sous le z/OS : DB2 & ARMIDE

DB2 est l'un des SGDB⁷ d'IBM. Il utilise tout comme MySQL le langage SQL. Sous TSO, je peux manipuler et accéder à des Bases de Données avec les utilitaires reconnus Platinum et Spufi. Ils me permettent de visualiser le contenu des tables par le biais de différentes commandes. Ils me permettent également d'écrire du code SQL afin de lancer des requêtes. Une légère différence entre les deux utilitaires est que Spufi va permettre de lancer des requêtes et nous créer un fichier qui contiendra la requête lancée par la même initiative. Ce cas échéant est utile lors de preuves de cas de test.

4.7.1 DB2 & COBOL

Comment l'interaction entre un programme COBOL et une Base de Données sous DB2 peut-elle s'effectuer ? Cette sous-section répondra à cette interrogation.

Les programmes COBOL interagissant avec des bases de données doivent utiliser des ordres SQL renfermés dans les mots clés EXEC SQL et END EXEC SQL. Ces instructions signifient qu'à partir de la ligne à laquelle commence l'EXEC SQL, du code SQL (SELECT, FETCH, INSERT...) doit être inséré. Les instructions SQL ont pour but d'aller requêter dynamiquement les tables DB2 précisées dans les instructions même du programme. Un seul programme COBOL peut contenir plusieurs EXEC SQL. Néanmoins, le compilateur COBOL n'a pas la capacité de lire et comprendre le SQL en tant que tel. Une opération en plus est nécessaire afin de préparer la bonne exécution d'un programme COBOL-DB2. Cette opération s'appelle un BIND. Un BIND permet d'associer les ordres SQL contenus dans un programme COBOL avec le programme lui-même une fois celui-ci compilé dans l'environnement de tests unitaires. Un programme COBOL compilé se nomme load module ou module source. Les ordres SQL sont traduits par la suite de leur côté et vérifiés.

Ensuite, il ne reste plus qu'à indiquer dans le JCL d'exécution la base de données avec laquelle nous souhaitons interagir puis l'utilitaire dédié à la communication avec DB2. La connexion réussie entre le load module et la table doit en découler. Si une différence apparaît, une erreur sortira.

4.7.2 ARMIDE : Un Base de Données de paramètres

Outre DB2, une autre base de données est présente chez la LBP : elle se nomme ARMIDE. Elle est propre à notre client et notre environnement de travail.

Son objectif est de contenir des informations de paramétrage, soit des informations qui ne seront pas amenées à être modifiées. L'impact est de réduire l'existence de certaines tables DB2 et donc de réduire le nombre de programme effectuant des accès aux tables car ces actions sont peu performantes.

4.8 JCL – Scripts et exécution de composants batch

⁷ Système de Gestion de Base de Données



JCL signifie Job Control Language. C'est un langage informatique de type script qui permet d'agencer les différentes étapes d'un traitement, faisant intervenir des programmes COBOL mais aussi un certain nombre d'utilitaires spécifiques.

4.8.1 Structure d'un JCL

J'ai appris au cours de mon alternance à CGI comment réaliser de a à z des scripts d'exécution de composant Mainframe (aussi appelés « jobs »). J'ai également appris comment automatiser et centraliser la création de ces derniers de manière à pouvoir les livrer avec les composants dans les environnements supérieurs afin que mes collègues puissent travailler avec sans avoir à écrire leur propre JCL.


```

1 //JCLEXTU JOB (XXXX,E,,500),'JCLTU',CLASS=X,MSGCLASS=X,
2 //      NOTIFY=&SYSUID,COND=(8,LE)
3 //*
4 //*****
5 //* Information sur le JCL ici
6 //*-----
7 //* Information sur la génération du JCL à placer en commentaires
8 //* Ajouter à cela le niveau d'urgence du traitement
9 //* Ajouter le nom de l'auteur
10 //*****
11 //* Description spécifique du rôle du JCL
12 //*****
13 //*
14 //*
15 //JOB LIB DD DSN=XXXXXXXXXXXXXXXXXX,DISP=SHR
16 //      DD DSN=XXXXXXXXXXXXXXXXXX,DISP=SHR
17 //      DD DSN=XXXXXXXXXXXXXXXXXX,DISP=SHR
18 //      DD DSN=XXXXXXXXXXXXXXXXXX,DISP=SHR
19 //      DD DSN=XXXXXXXXXXXXXXXXXX,DISP=SHR
20 //      DD DSN=XXXXXXXXXXXXXXXXXX,DISP=SHR
21 //      DD DSN=XXXXXXXXXXXXXXXXXX,DISP=SHR
22 //      DD DSN=DB2.XXXXXXXXXXXXXXXXXX,DISP=SHR
23 //      DD DSN=XXXXXXXXXXXXXXXXXX,DISP=SHR
24 //*
25 //*
26 //*****
27 //DEBJCL EXEC PGM=IDCAMS
28 //*****
29 //*COM-DESC=STEP DE DELETE AUTOMATIQUE EN DEBUT DE JOB
30 //*
31 //SYSPRINT DD SYSOUT=*
32 //SYSIN DD *
33 DELETE XXXXXXXXXXXXXXXXXXXX.FICHER
34 DELETE XXXXXXXXXXXXXXXXXXXX.FICHER
35 DELETE XXXXXXXXXXXXXXXXXXXX.FICHER
36 DELETE XXXXXXXXXXXXXXXXXXXX.FICHER
37 SET MAXCC = 0
38 //*
39 //*
40 //*****

```

Figure 5- Structure type d'un script d'exécution ou JCL

Un JCL comme tout type de script s'écrit et se lance de manière séquentielle, c'est-à-dire que chaque instruction va être exécutée en partant de la première du haut et l'une après l'autre. Le script utilise des cartes pour atteindre cet objectif.

Il est séparé en plusieurs étapes :

- Lignes 1 et 2 : une carte obligatoire appelée JOB est écrite. Cette carte explique au script que l'on instancie un JCL.

- Lignes 15 à 23 : une carte obligatoire appelée JOBLIB vient spécifier les librairies dans lesquelles je dois retrouver mon programme.
- Des lignes 27 à 38 j'effectue une étape de « DELETE préventif ». Une bonne pratique est de vérifier au début de chaque script d'exécution que les fichiers qui sont sur le point d'être créés n'existent pas déjà, auquel cas les supprimer sera nécessaire.
- Les lignes suivantes, non visible sur la figure 5, vont exécuter un programme ciblé en utilisant une carte EXEC réservée à cet effet. EXEC a pour fonction d'exécuter un programme ou utilitaire IBM.

Tous les JCLs ont la même structure. Il est possible de conditionner des scripts avec des instructions IF/ENDIF.

4.9 Organigramme des traitements sous SIROCCO

Le schéma ci-dessous représente un schéma typique d'un programme lié à la compréhension d'un dossier de conception détaillée et d'une demande formulée. Ce programme peut être codé en COBOL, comme en PacBase ou PTF, les règles s'appliquent pour tout type de langage.

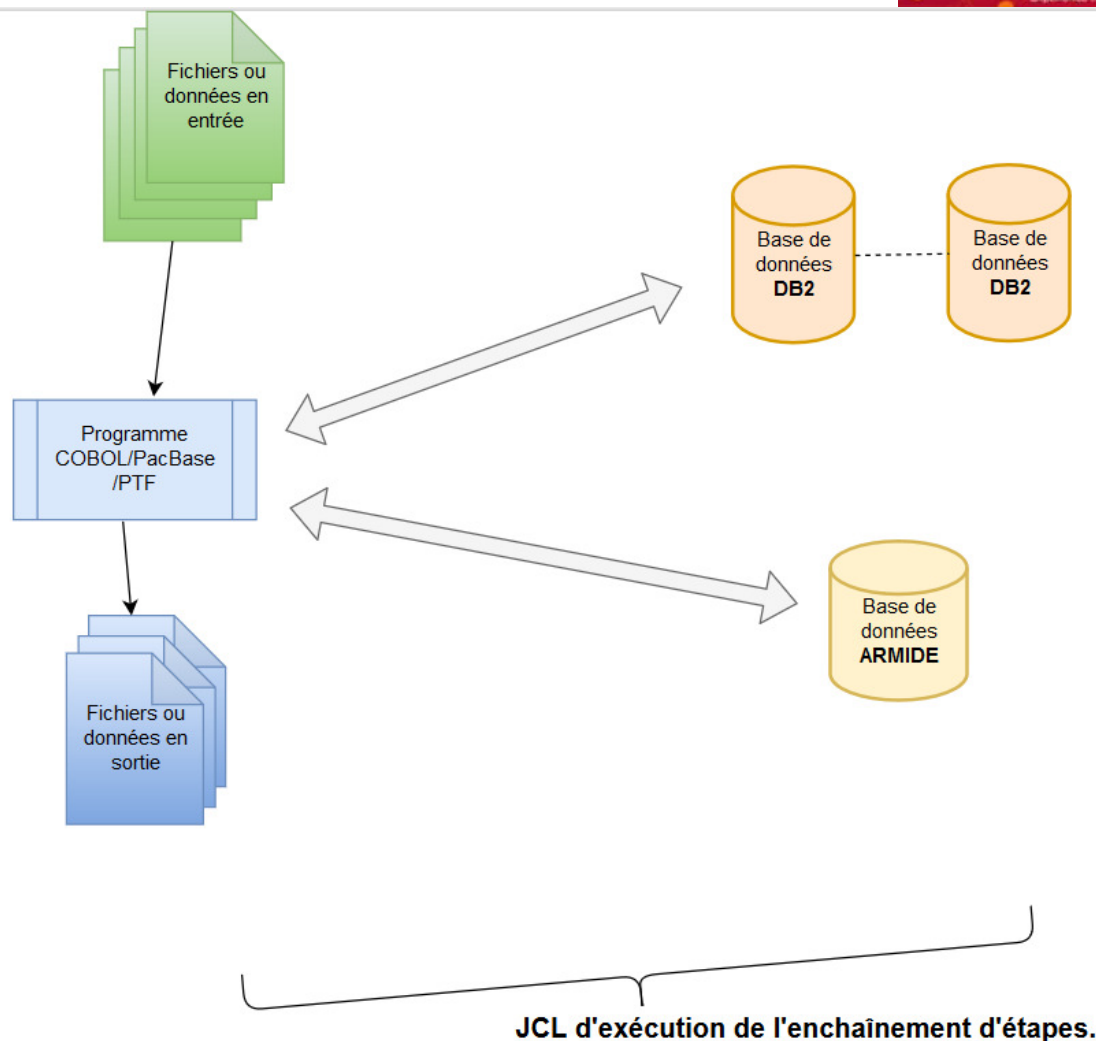


Figure 6 - Schéma organisationnel de traitement avec un programme et fichiers

Au cours de l'année passée au sein de CGI, j'ai appris à maîtriser les interactions des différents composants présent dans ce schéma. Un programme COBOL va créer des fichiers à l'issue d'un traitement réalisé, et peut avoir besoin d'interagir avec des bases de données DB2 ou ARMIDE (Figure 6).

De manière générale, lorsque nous recevons une demande client spécifique, cette demande ou expression de besoin du métier est traduite par la Maitrise d'Ouvrage (MOA) en une Définition Fonctionnelle du Besoin (DFB), pour ensuite être analysée par la Maitrise d'œuvre (MOE). Un analyse MOE traduira cette DFB en un document de conception générale (DCG), spécifiant les règles FONCTIONNELLES à mettre en œuvre ou à modifier. Les règles de gestion fonctionnelles doivent encore être traduites en un document de conception technique « détaillée » (DCOD ou DCD) contenant cette fois-ci les détails techniques de la réalisation du composant

Une fois le DCOD rédigé, c'est le développeur qui doit analyser le détail écrit dans ce document afin de réaliser un programme répondant à ses exigences. C'est la réalisation du programme qui aboutit à la traduction du DCOD dans le langage de programmation désiré. C'est cette étape du développement d'un programme que j'ai le plus souvent effectué. Dans un DCOD figure souvent un organigramme ressemblant au schéma ci-dessus. En effet, le programme codé n'est bien souvent

qu'une petite partie de la chaîne de traitement. Il est donc nécessaire de décrire tout ce qui va encadrer notre futur programme, ou notre programme à modifier en tant que tel, afin de récupérer les bons fichiers et de les positionner en entrée dans notre script d'exécution, ainsi que les fichiers de sorties qui seront créés à l'issue du traitement réalisé par le programme COBOL.

Chaque programme est le résultat d'une demande formulée par notre client. Notre analyste réalise le DCOD et nous informe dans celui-ci des différents processus à mettre en place : les fichiers à collecter afin de les mettre en entrée, dans notre cas cela sera par exemple un fichier contenant des données clients, date de début de contrat, date de fin de contrat etc... L'analyste doit également nous donner le nommage des fichiers à créer en sortie. Nous écrivons ces noms dans le JCL lors d'exécution du programme.

Mon rôle en tant que développeur est donc de réaliser le programme en respectant parfaitement, tout en gardant toujours un regard critique, les algorithmes décrits dans le DCOD. Une fois écrit, un programme COBOL doit être compilé pour pouvoir être exécuté. Compiler un programme dans le mainframe revient à créer un « source » (tout comme en C par exemple), afin que le programme soit considéré comme exécutable par le système. J'ai mentionné le nom complet de load module dans la sous-section précédente. Je dois après l'envoyer dans un gestionnaire de packages propice à l'obtention de ce statut : il s'agit d'ENDEAVOR. ENDEAVOR peut être comparé au système de « versioning » délivré par Git ou SVN.

Compiler un programme, consiste traduire le code COBOL en code source compilé et exécutable (par le Mainframe, en l'occurrence). C'est une tâche récurrente que tout programmeur est amené à mener à bien.

En résumé : les tâches que je réitère lors de demande sont :

- 1) Lecture et compréhension du DCOD (dans le cas où je ne suis pas rédacteur)
- 2) Préparation d'une fiche de tests unitaires
- 3) Ecriture du code et compilation
- 4) Préparation des fichiers physiques d'entrée et sortie du programme
- 5) Exécution avec un JCL contenant les indications pointant vers les fichiers et lançant le source
- 6) Vérification des cas de test.

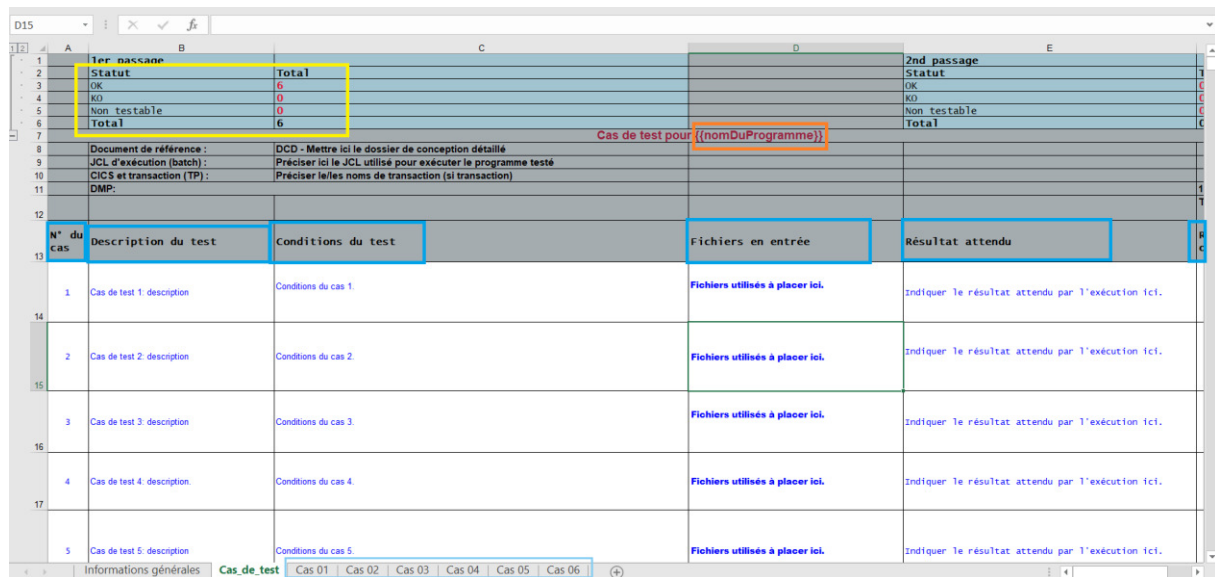
4.10 Tester un composant

Une fois un programme compilé puis exécuté, vient la partie la plus consommatrice en terme de temps de travail : la réalisation de tests unitaires⁸ (RTU). Réaliser des tests unitaires consiste à tester cas par cas chaque infime fonctionnalité du programme développé. L'objectif est avant tout de vérifier que le programme répond aux exigences demandées puis de rédiger une fiche détaillant chaque mise en œuvre des cas, afin de la livrer à l'analyste qui se chargera d'effectuer des Tests d'Assemblage (TA).

⁸ RTU englobe à la fois le développement d'un composant et ses tests unitaires. J'emploie la notion de RTU ici mais le paragraphe fait surtout référence aux déroulements des tests unitaires.

Les TU se réalisent avec le logiciel Microsoft Word ou Excel en fonction du nombre de cas de tests que nous avons à opérer. Si mes cas sont nombreux, j'ai recours à Microsoft Excel en raison de sa capacité à séparer les feuilles de calculs (255 feuilles maximum par fichier) et à gérer les références croisées.

Afin de réaliser mes cas de test je dois avoir à disposition un JCL pouvant exécuter mon programme. Deux cas de figure s'offre à moi : je code à la main le JCL ou bien j'utilise un outil fourni par notre client afin de générer automatiquement un JCL prêt pour exécuter mon programme. Cet outil se nomme eDMP⁹.



N° du cas	Description du test	Conditions du test	Fichiers en entrée	Résultat attendu
1	Cas de test 1 description	Conditions du cas 1.	Fichiers utilisés à placer ici.	Indiquer le résultat attendu par l'exécution ici.
2	Cas de test 2 description	Conditions du cas 2.	Fichiers utilisés à placer ici.	Indiquer le résultat attendu par l'exécution ici.
3	Cas de test 3 description	Conditions du cas 3.	Fichiers utilisés à placer ici.	Indiquer le résultat attendu par l'exécution ici.
4	Cas de test 4 description	Conditions du cas 4.	Fichiers utilisés à placer ici.	Indiquer le résultat attendu par l'exécution ici.
5	Cas de test 5 description	Conditions du cas 5.	Fichiers utilisés à placer ici.	Indiquer le résultat attendu par l'exécution ici.

Figure 7- Exemple type de fiche TU avec Microsoft Excel

L'image ci-dessus montre un exemple type de fiche de tests unitaires. Les colonnes ont été généralisées.

Le processus se déroule par le biais de plusieurs étapes d'écritures :

- 1) A gauche figure le numéro correspondant à chaque cas de test. Pour le cas numéro 1, j'inscris 1, pour le cas 2 j'inscris 2 et ainsi de suite
- 2) Ensuite, sur la colonne « Conditions du test », j'écris les conditions impliquant mon cas de test.
- 3) La colonne « Fichiers en entrée » est l'emplacement où je spécifie les noms physiques des fichiers que je passe en entrée du test de mon programme.
- 4) La colonne « Résultat attendu » exprime les alimentations que j'attends dans mes fichiers de sortie. Elle exprime aussi les codes retour ou statut d'exécution du programme attendus. La compréhension de l'algorithme est reflétée dans cet emplacement.
- 5) Une colonne nommée « Résultat obtenu » à peine visible sur la figure 7, contient un lien hypertexte pointant sur la feuille du cas de test concerné. Par exemple pour le cas de test 1, je clique sur le lien hypertexte Cas 01 et serai redirigé vers la feuille Cas 01 présente en bas de la figure 7. Cette opération est renouvelable pour chaque cas.

⁹ DMP signifie Dossier de Mise en Production. Il s'agit d'un Package que l'on réalise qui va contenir tous les éléments liés à notre besoin. Ce package va être livré à une équipe opérant en production.

- 6) Sur une colonne intitulée « Statut » non-visible sur la figure 7, doit se trouver un label expliquant le statut retourné par le cas de test. Trois statuts sont disponibles : OK pour un cas qui s'est déroulé correctement, KO pour un cas devenu bloquant et NON-TESTABLE pour un cas finalement impossible à tester.
- 7) Enfin, une colonne non-visible situé sur l'extrême droite de l'image précise la date exacte en format JJ/MM/SSAA du lancement du cas de test. Cette information est importante car elle certifie le moment précis et daté du lancement du cas.

La fiche de TU est un livrable. Une fois rédigée et validée par mon analyste, je la mets à disposition et elle sera envoyée avec la livraison du composant testé ainsi que tous les autres documents nécessaires à la demande exigée.

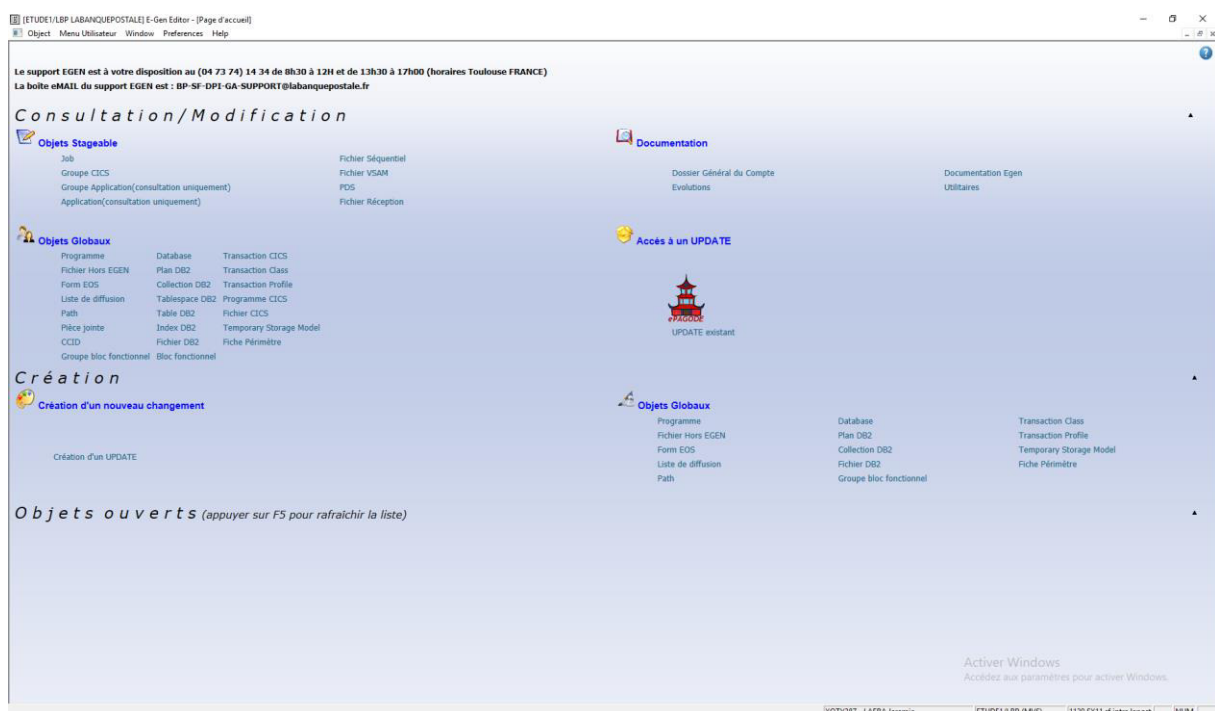


Figure 8- Interface d'accueil de l'outil eDMP

La figure 8 montre l'interface d'accueil de l'outil eDMP. Ce dernier a pour objectif d'automatiser les processus d'exécution de programmes ou de scripts mainframes. Il nous permet de faire du déploiement de composant.

Partie 5

5. Compétences acquises au cours de l'année en Alternance U'DEV2

Cette partie va énumérer, en plus des compétences explicitées de la partie 4, les compétences que j'ai obtenues au cours de l'année passée.

5.1 Compétences acquises au sein de CGI Le Haillan

5.1.1 Analyse, Conception et développement de programmes et composants mainframe

Au cours de l'année d'apprentissage, j'ai pu concevoir et développer plusieurs composants Mainframe, essentiellement des composants batch, mais il a pu m'arriver de modifier certains composants transactionnels de type SA.

J'ai donc été amené à m'initier à la rédaction de Dossiers de Conception Détaillée (DCOD) simples, en vue de les délivrer par la suite à des collègues développeurs. La suite de ma progression à moyen terme dans le cadre d'un CDS Référentiel consistera notamment à me familiariser davantage avec l'écriture des DCOD et d'acquérir plus de connaissances fonctionnelles afin de rédiger des Dossiers de Conception Générale (DCG).

5.1.2 Evoluer un programme existant : modifier un algorithme sans générer de dysfonctionnement

Modifier un composant n'est pas chose rare. Une modification implique une nouvelle version d'un DCOD. Par conséquent, à chaque nouvelle demande du client sur un composant existant, il est indispensable de modifier les documents de spécifications fonctionnelles et techniques. Ce sont des tâches que j'ai effectuées chez SIROCCO. Elles sont divisibles par deux.

En premier lieu, je dois récupérer la version actuelle du DCOD et la mettre à jour. J'utilise les outils fournis par la suite Office et plus particulièrement Microsoft Word et son outil de Révision. Ainsi, les collaborateurs pourront apercevoir de manière précise les évolutions que j'ai apportées au document. Je modifie les algorithmes dans le DCOD et je fais en sorte que les évolutions ne vont pas impacter les fonctionnalités actuelles du programme.

Puis, je commence les modifications sur le programme, après avoir pris le soin d'écrire ma nouvelle fiche de tests unitaires. La subtilité dans cette étape réside dans l'ajout de tests de non-régression ou TNR. En effet, dès qu'une évolution sur un programme survient, des TNR sont de rigueur. Ces tests visent à vérifier que les évolutions n'ont pas impacté ou apporté d'anomalie(s) à l'algorithme des traitements du programme actuel. Une technique utile que j'emploie au sein de SIROCCO consiste à prendre mon JCL d'exécution et le lancer deux fois : une fois avec la version actuelle du programme

dans l'environnement dans lequel il se situe, et une deuxième fois dans l'environnement de tests unitaires sous sa nouvelle version modifiée. En lançant le JCL à deux reprises, je change également mes noms de fichiers et données en sortie afin d'avoir deux versions. Une fois cela fait, je réalise une comparaison entre les deux fichiers générés. Pour ce faire, j'ai recours à l'utilitaire File-Aid qui est un manipulateur de fichiers.

5.1.3 Débogage de programme mainframe

Une des tâches les plus indispensables que j'ai dû m'accaparer est le débogage de programmes mainframe.

J'ai eu recours à l'utilitaire Xpediter mais également à l'outil de débogage embarqué dans UDC qui s'inspire fortement du débogueur Eclipse. Xpediter est complet : il possède des commandes pour pouvoir par exemple afficher les valeurs des variables en à un instant T. Il permet également de placer des conditions (commande WHEN) dans le code au moment du débogage de manière à aller directement sur un cas précis voulu. L'outil débogueur d'UDC est également complet : possédant à peu près les mêmes fonctionnalités qu'Xpediter, il offre en prime une interface graphique dite user-friendly¹⁰.

L'action de déboguer un programme m'a permis à de nombreuses reprises du trouver des anomalies dans mes cas de tests, d'en alerter mes collaborateurs ou bien de résoudre des cas difficiles d'erreurs d'écriture dans mes fichiers de sortie.

5.1.4 Rédaction de tests unitaires et tests d'assemblage

Comme évoqué dans la partie 4, réaliser les Tests Unitaires est la dernière étape avant la livraison et le déploiement de package contenant les fichiers et JCL liés à mon composant.

Réaliser des tests unitaires consomme en moyenne environ 70% des charges. Une bonne écriture, un bon français ainsi qu'un bon esprit critique sont requis pour rédiger des fiches TU compréhensibles pour tous.

Le processus est le suivant : une fois le programme compilé puis exécuté, je crée une feuille Excel par cas de figure, et chaque cas doit contenir les informations concernant l'algorithme à tester et les fichiers utilisés pour le tester. Chaque cas de test possède un fichier différent. De cette manière, je conserve une trace de chacun de mes tests et laisse mes fichiers à disposition pour que l'analyste en charge des tests d'assemblage (TA) puisse aisément récupérer mes données.

J'ai eu à quelques reprises l'opportunité de produire des TA. C'est le cas lorsque je travaille sur des Services Applicatifs (SA). Les TA sont plus rapides à effectuer pour les SA et j'ai été habilité à en pratiquer dans le cadre d'évolutions. Néanmoins, j'ai dans la plupart de cas travaillé avec des programmes batch.

La phase de tests, que ce soient des tests unitaires ou non représente la notion de travail d'équipe car des échanges réguliers ont lieu entre les développeurs et analystes.

¹⁰ Se dit d'une application ou outil facile d'utilisation.

5.1.4 Mise à disposition de composant(s) ou programme(s) vers les environnements supérieurs – Déploiement

Ultime phase de ma tâche de RTU en mainframe est la mise à disposition de tous les objets que j'ai dû créer ou générer lors de mes cas de test.

À ma disposition se trouve ENDEAVOR pour livrer mes programmes dans les environnements supérieurs, mais également eDMP l'outil interne qui me permet de créer des packages contenant le JCL officiel d'exécution du programme ainsi que les fichiers et autres entités gravitant autour de ce dernier.

La mise à disposition de ce package, en plus de monter le programme sous ENDEAVOR et de livrer ma fiche de TU contribuent à une phase qualifiée de déploiement.

5.1.5 Estimer mon Reste à faire (RAF) et contrôler les délais

L'estimation du Reste à Faire ou RAF est très importante. Elle permet de tenir au courant les collègues avec lesquels je travaille afin qu'ils puissent mettre à jour leur planning.

J'ai à ma disposition des outils internes me permettant de m'imputer sur les demandes. Le bon comportement à avoir est celui de s'imputer deux fois par jours de travail : une fois aux alentours de midi et une fois avant de terminer la journée. Ainsi, les analystes et responsables applications peuvent se tenir à jour en consultant mes imputations depuis l'interface de nos outils internes.

5.1.6 Communiquer avec mon équipe et mettre à jour mes analystes sur mes avancées ou blocages

La communication avec mon équipe est un facteur essentiel à mon métier de développeur. Lors de blocage ou point incohérent rencontré dans les spécifications ou programmes, je m'adresse directement à l'analyste qui est assigné à la demande. Les communications se font par le biais d'emails ou appels téléphoniques si je travaille avec un collègue situé à distance. La communication dans l'informatique et à CGI octroie une productivité accrue. Elle va en effet accentuer la vitesse d'exécution car c'est l'interaction qui aide le développeur à résoudre certains problèmes.

5.1.7 Concevoir des packages contenant des solutions logicielles automatisées

L'outil eDMP un l'un des outils auquel j'ai le plus souvent à faire. Les tâches les plus récurrentes que je dois réaliser avec ce dernier me permettent d'automatiser des traitements, de générer des JCLs qui pourront être utilisés de manière indépendante par des collègues travaillant dans d'autres environnements. C'est un gain de temps considérable puisque cela nous évite de tous écrire le même JCL à la main à chaque fois qu'un programme monte d'environnement.

5.1.8 Livrer un logiciel conforme aux besoins du client

La livraison d'un logiciel conforme aux besoins du client est un aspect fondamental du métier de développeur et d'analyste. Il est donc nécessaire de fixer des objectifs faisables avec le client dès le début d'une demande. À partir de ce postulat, il est de mon devoir de réaliser à la lettre près le composant exigé avec les fonctionnalités qu'il doit contenir.

À plusieurs reprises j'ai pu délivrer un composant ou une suite de composants qui s'inscrivent dans une conformité verrouillée des besoins du client. Ces composants étaient accompagnés de fiches de

tests unitaires détaillant les tests élaborés sur les différentes fonctionnalités à implémenter. Les composants voyagent par la suite à travers le gestionnaire ENDEAVOR jusqu'à arriver en environnement de production.

5.1.9 Respecter des contraintes et conventions de qualité de code en termes d'architecture logicielle

À CGI, j'ai été amené à respecter des contraintes et conventions de qualité de code. Le COBOL possède certaines instructions consommatrices et donc peu performantes. Le langage va aussi mettre davantage de temps CPU à compiler lorsque de nombreuses conditions IF/END-IF s'enchaînent par exemple.

Afin de limiter ces longs temps de compilation, un plan qualité a été mis en place. Pour les programmes PacBase, il se note PQC pour Plan Qualité de Code. Quant aux programmes COBOL et PTF, il s'agit d'une note de qualité. Je dois respecter les normes de manière à ne pas tomber sous un certain seuil (note). Par conséquent, lors de compte-rendu de compilation, si j'utilise des critères visant à baisser cette note de qualité, alors le compilateur me le signalera sous forme de message Warning en fin de compte-rendu.

Je dois donc soit déployer des stratégies différentes afin de supprimer ces Warning et respecter les normes de qualités en vigueur, soit fournir une explication justifiée à mon analyste lors de la livraison du composant. Dans le premier cas, je peux avoir recours à un EVALUATE en COBOL qui est l'équivalent du switch. Cela me permet de contourner l'utilisation de multiples IF/END-IF. Il existe d'autres cas de figure où je dois être amené à vérifier mon code en termes de respect des plans qualités.

5.1.10 Clôturer une mission

Au cours de l'année passée à CGI, j'ai appris à exécuter une mission de bout en bout. De la conception à la réalisation des tests unitaires allant jusqu'à la livraison de la fiche de TU, j'ai su réaliser chacune des tâches attendues par les analystes et responsables applications.

À chaque fin de demande, je me charge de notifier l'analyste qui sera en charge d'effectuer les TA derrière mes TU. Il peut, à ce moment-là préparer son planning afin de réserver un créneau. Je notifie par la même occasion la responsable applications en charge de la gestion de la demande afin que cette personne elle aussi puisse mettre à jour son planning. Dans un cas sur deux, je travaille avec l'équipe Nantaise. Il est par conséquent extrêmement crucial que je communique avec l'équipe lors de la fin de mission, soit par mail, soit par téléphone.

Enfin, une fois ces deux actions derrière moi, je m'impute sur cette tâche une ultime fois, et le total des charges consommées apparaît. Cela me permet ensuite d'effectuer une rétrospective et de constater les écarts constatés ou non avec les charges initialement prévues.

5.2 Compétences acquises à l'extérieur de CGI (EPSI, Projets Personnels)

Cette section s'articule autour d'autres notions indispensables au métier de développeur. Elle va traiter du paradigme de la programmation orientée objet, de logiciels partageables, d'architecture multicouches etc...

5.2.1 Conception de logiciels bénéficiant d'une architecture

N'Tiers

J'ai été amené que ce soit à l'école EPSI, ou comme en projet personnel à concevoir des logiciels multicouches. Afin d'illustrer ce chapitre, je parlerai à la fois d'architecture N'Tiers mais également de programmation orientée objet (POO ou OOP en anglais) parce que de nos jours un nombre importants d'applications bénéficiant d'une architecture possédant plus de deux couches sont écrits en langages OOP.

5.2.2 Développement de programmes utilisant la Programmation Orientée Objet (OOP) – Exemple d'un blog pour Entreprises Sociales

À l'école EPSI et à la maison, j'ai pu pratiquer et développer des compétences en programmation orientée objet, dénotée POO ou OOP en anglais. Je vais démontrer les compétences accumulées en OOP en illustrant des exemples et extraits de codes provenant d'une application réalisée hors EPSI et CGI. Il s'agit d'une application de publication d'articles (blogs) pour Entreprise Sociale ou toute société désirant s'accaparer une espace social privatif.

Une entreprise sociale en quelques mots est une entreprise ayant pour but de répondre à des besoins sociaux mal couverts par l'Etat et le secteur privé.

Pour réaliser mon système de publication d'articles, j'ai choisi d'avoir recours au Framework de développement web Flask.

5.2.2.1 Flask

Pour débiter et développer une application web simple et en vitesse, le Framework de développement web Flask est adapté. Flask est un Framework Python très léger d'installation puisqu'il ne contient pas de librairie externe¹¹ par défaut lors de son installation. Il appartient au développeur d'ajouter des « extensions » en fonction de son besoin en les installant au fur et à mesure. Différentes extensions existent à l'heure actuelle comme un gestionnaire de formulaire, un crypto de mot de passe contenant des fonctions de hachage, un gestionnaire de base de données etc... Flask incorpore seulement un noyau de fonctionnalités et un système de génération de templates appelé Jinja lors de son installation par défaut. Pour toutes ces raisons, Flask est considéré comme un **microframework**.

Installer une librairie permet d'utiliser les fonctionnalités qu'elle transporte : cela peut être un ensemble de méthodes ou classes.

5.2.2.1.1 Règles de gestion du projet AltarisCompanyBlog

Les règles de gestion sont les suivantes :

¹¹ Ou librairie Tiers (Third-Party Library en anglais). C'est un logiciel tiers, développé par une entité afin d'être utilisable ou vendu. Ce logiciel doit être réutilisable.

Un internaute non connecté peut voir la liste des postes publiés. Un utilisateur c'est-à-dire un internaute étant inscrit et connecté peut voir la liste des postes publiés, ajouter un poste et supprimer ces propres postes uniquement. Un utilisateur ne peut pas supprimer le poste d'un autre utilisateur. Un utilisateur peut avoir une photo de profil qu'il peut téléverser directement sur l'application.

Les évolutions à venir seront l'instauration d'un système administrateur ainsi que l'amélioration de l'interface utilisateur.

5.2.2.1.2 Architecture

L'architecture d'une application développée en Flask est représentée comme suit :

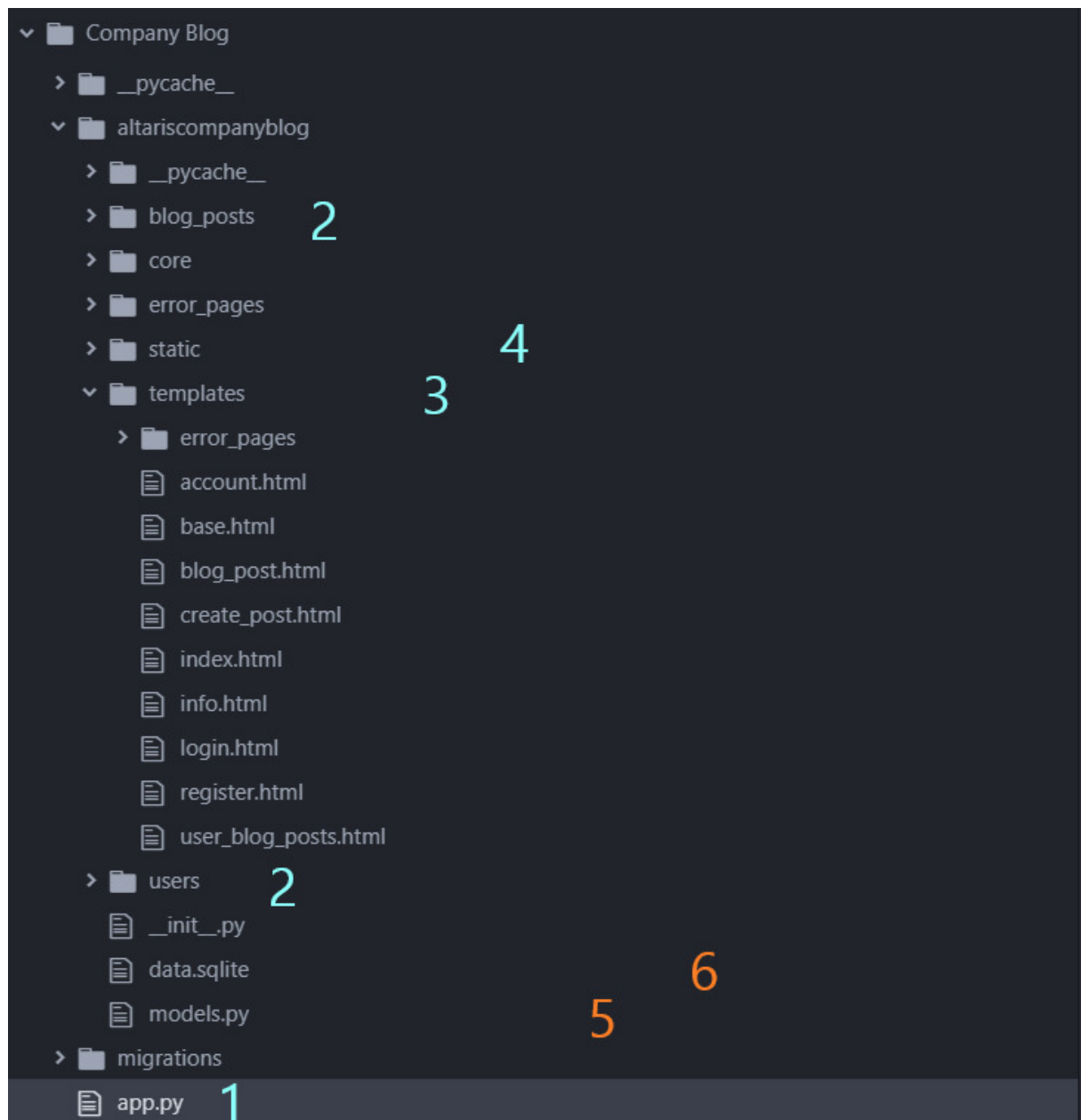


Figure 9- Architecture d'un projet codé en Flask



La figure 7 illustre l'organisation des répertoires à élaborer lors de la création d'un projet d'une certaine taille. Dans le cadre de la réalisation du projet AltarisCompanyBlog, j'ai dû séparer mon code dans différents répertoires.

- Le fichier *app.py* (numéro 1) est le fichier maître de l'application. C'est le fichier que nous allons exécuter afin de lancer l'application. Il doit impérativement se situer à la racine de notre application. Il contient les informations relatives à tous les répertoires présents dans celui appelé *altariscompanyblog*, et contient l'instruction de lancement de l'application.
- Les répertoires *blog_posts* et *users* (numéro 2) sont les entités respectives du projet.
- Les « *templates* » (numéro 3) représentent les vues de l'application et par conséquent la couche de présentation. En l'occurrence, ce sont des fichiers HTML étant donné que le front sera géré par du HTML accompagné de Bootstrap.
- Le dossier *static* (numéro 4) doit contenir tous les fichiers de type CSS et JavaScript mais également les images ou vidéos stockées.
- En numéro 5 figure le fichier *models.py* qui se charge d'instancier les tables de notre base de données. C'est la définition de la couche d'accès aux données.
- En numéro 6 figure un fichier intitulé *data.sqlite*, qui est généré lors de la création de la base de données sous SQLAlchemy utilisé dans mon application.
- Il est à noter qu'un fichier *__init__.py* se situe au même niveau que *models.py*. Ce fichier a pour rôle de créer les instances de notre application et base de données. Il effectue également les sections de mon application sous forme de **blueprints**. Les blueprints sous Flask sont des patrons qui permettent de définir des entités à partir de sections. On peut imaginer ces patrons comme des empreintes que l'on aurait réalisées et imprimées dans des moules. On peut alors appliquer ces empreintes à différents endroits de l'application. J'utilise cette notion pour appeler mes entités User et BlogPost depuis mon fichier *__init__.py* et les référencer dans les autres fichiers.

L'architecture d'un projet Flask, ressemble en tout point à une architecture de type MVC¹². Le MVC est un design pattern utilisé dans la plupart des applications web actuellement. Il est également convoité dans les applications Android sous l'appellation MVP¹³. Cependant, ce patron de conception date d'avant les premières créations de sites et applications web. Il était implémenté dans les applications de client lourd c'est-à-dire les applications fenêtrées comme par exemple Photoshop, Sony Vegas ou Word.

5.2.2.1.3 Interface et affichage de l'application

Grâce au système de gestion de templates fourni par Jinja (Flask) je peux obtenir un affichage rapide, stylisé avec Bootstrap (que j'ai ajouté) et responsive :

¹² Model-View-Controller

¹³ Model-View-Presenter, une variable du modèle MVC couramment employée

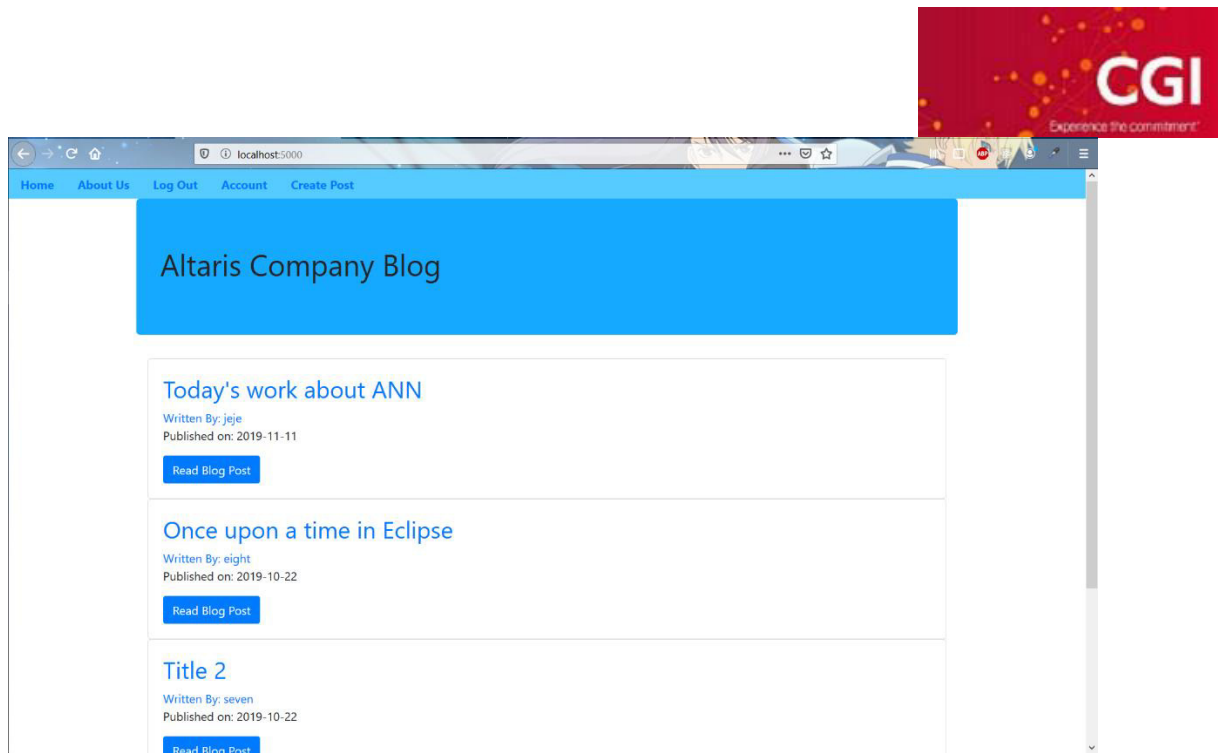


Figure 10 - Page d'accueil Altaris Company Blog avec liste de postes d'affichée (l'utilisateur est connecté)

L'application possède un design adapté pour fonctionner sur smartphone ou tablette :

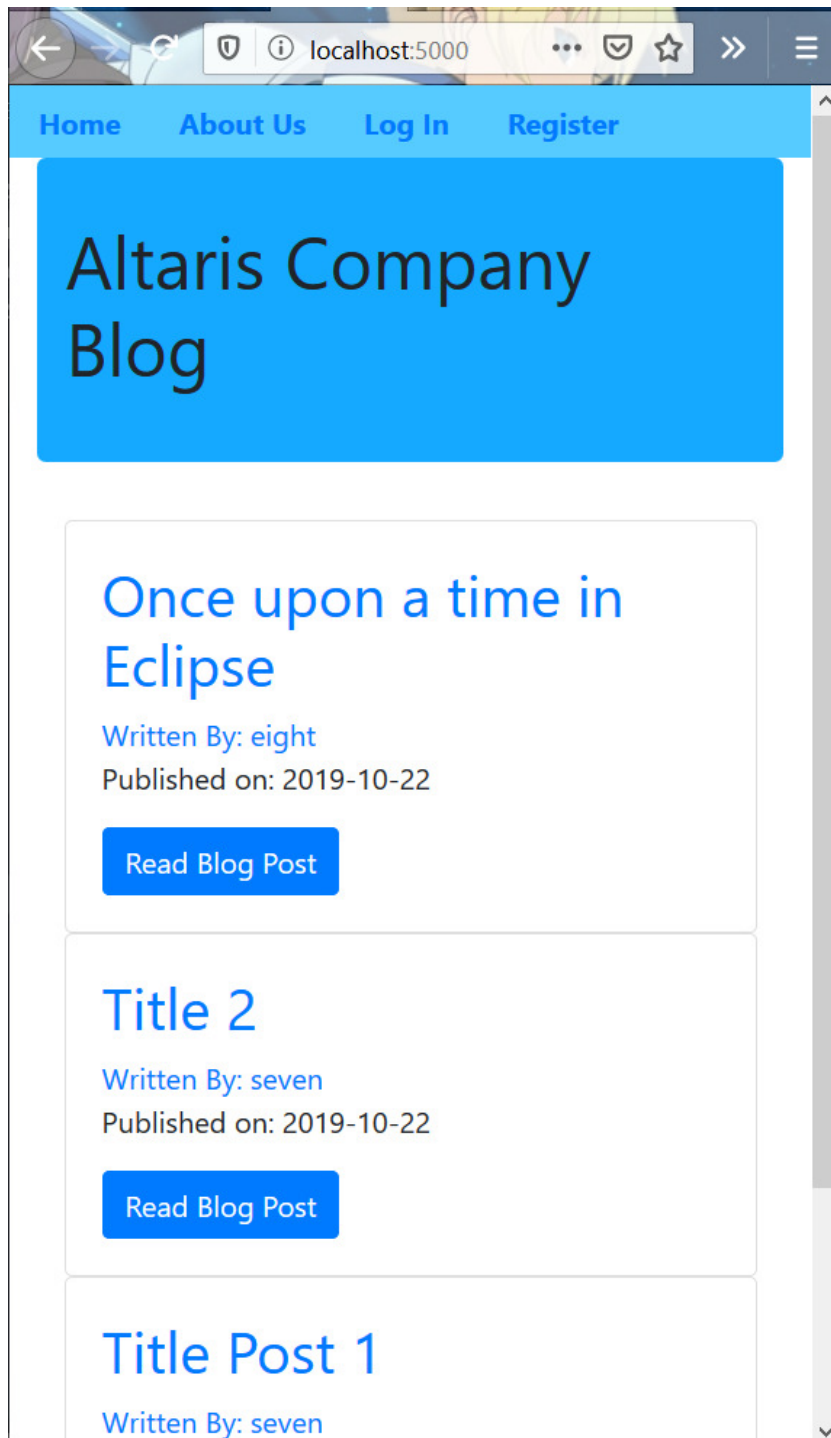


Figure 11 - Ecran d'accueil affiché pour téléphone avec une liste des postes publiés en pile

Sur l'écran affichant la liste des postes publiés, on peut apercevoir les éléments classiques d'une structure de pages HTML. Néanmoins, quelque chose d'intéressant se produit dans le code :

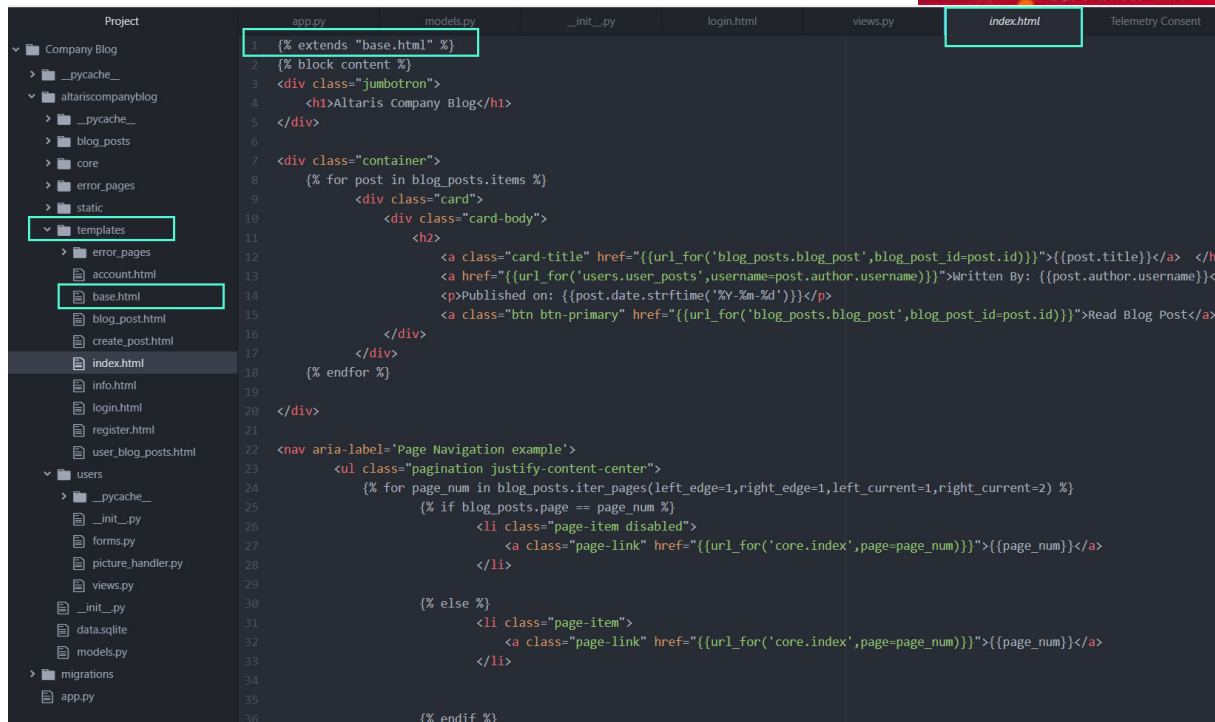


Figure 12 - Template HTML héritant d'une base

Le fichier présent (figure 12) correspond aux affichages des figures 10 et 11. Il n'y pas tout le code HTML car celui-ci est placé dans un fichier *base.html*. En utilisant une fonction *extends* (ligne 1), je permets à mon fichier *index.html* d'hériter du contenu de *base.html*. Jinja me donne accès à cette fonctionnalité. Grâce à ce système de templates, j'épouse le principe du DRY¹⁴.

5.2.2.2 La Programmation Orienté Objet en théorie

En ce qui concerne une programmation orientée objet de qualité, j'ai également choisi de m'adapter du mieux que possible aux principes de SOLID¹⁵.

Cet acronyme, apparu au début des années 2000, est tout d'abord imaginé par Michael Feather puis distillé et officiellement mis au point par Robert C. Martin. Ce dernier est notamment connu dans le domaine de l'informatique par son œuvre *Clean Code* mais aussi *Clean Architecture* dans lequel il mentionne Michael Feather énonçant ceci : « Aux alentours de l'année 2004, Michael m'envoya un email en me disant que si on réarrangeait les principes, cela donnerait l'acronyme SOLID; et c'est ainsi que les principes SOLID sont nés. ». Cette traduction est apportée par moi-même.

SOLID se fonde sur 5 principes que tout programme écrit avec un langage orienté objet doit tenter d'incorporer :

- **Single-responsibility** : chaque class codée ne devrait avoir qu'une seule responsabilité et donc un seul type de tâche ou service à effectuer.
- **Open / Close** : toute classe doit être ouverte à une mise à jour ou extension, mais doit être fermée à toute modification. Ce principe a pour but de réduire la régression suite à une évolution apportée.

¹⁴ Don't Repeat Yourself.

¹⁵ SOLID: Single Responsibility, Open / Close, Liskov Substitution, Interface Segregation et Dependency Inversion.

- **Liskov Substitution** : Barbara Liskov est une informaticienne célèbre (médaille John von Neumann, Prix Turing) et donne son nom à ce principe. Ce dernier se focalise sur le sous-typage. L'utilisation de méthodes abstraites est préconisée afin que des objets puisse les implémenter et les utiliser de manière variable.
- **Interface Segregation** : ce principe vise à prioriser la création de petites interfaces plutôt qu'une interface contenant de nombreuses déclarations et méthodes. Il vise à faciliter une décomposition du code, et de ce fait les tests unitaires notamment pour les développeurs Java.
- **Dependency Inversion** : ce principe vise à inverser les dépendances entre le modules et applications. Les projets peuvent pratiquer de l'injection de dépendance.

Au cours de l'année j'ai effectué diverses pratiques visant à me rapprocher de certains des principes SOLID. Il est à noter que certains langages informatiques sont favorables à ceux-ci. C'est le cas du Java étant donné qu'il possède des mots réservés **Interface** et **abstract**. La Java gère très bien l'abstraction et facilite la vie du développeur lors d'application de la **substitution de Liskov** par exemple.

La programmation orientée objet peut être interprétée avec le Langage de Modélisation Unifié (ou UML). Pour l'application AltarisCompanyBlog, des diagrammes de classes ont été réalisés afin de représenter les objets Utilisateurs (User).

5.2.3 Garantir un accès sécurisé aux données – Exemple illustré avec le blog pour Entreprise Sociale et Flask

Le projet de réalisation d'un blog pour Entreprise social est subdivisé en plusieurs sous-couches. L'une d'entre-elles est la couche d'accès aux données. La langage Java fait référence à cette couche en apportant la notion de DAO¹⁶. Ce pattern se charge de connecter nos tables aux bases de données avec des objets Java. Avec Flask, j'ai créé un fichier *models.py*, se chargeant de cette même tâche.

Une fois notre base de données et notre table définis conceptuellement parlant, il est possible de créer ce fichier afin d'y écrire les définitions de nos modèles :

¹⁶ Data Access Object

```
Company Blog > altariscompanyblog > models.py > ...
1
2 #user models and blog post model
3 #UserMixin -> allows us to user is authenticated, is allowed etc...
4 from altariscompanyblog import db,login_manager
5 from werkzeug.security import generate_password_hash,check_password_hash
6 from flask_login import UserMixin
7 from datetime import datetime
8
9 # setting up the user model and authentication
10 @login_manager.user_loader
11 def load_user(user_id):
12     return User.query.get(user_id)
13
14 class User(db.Model,UserMixin):
15     __tablename__ = 'users'
16
17     #primary key, unique and index are SQL constraints
18     id = db.Column(db.Integer,primary_key=True)
19     #default image as default since users must have an image
20     profile_image = db.Column(db.String(64),nullable=False,default='default_profile.png')
21     email = db.Column(db.String(64),unique=True,index=True)
22     username = db.Column(db.String(64),unique=True,index=True)
23     password_hash = db.Column(db.String(128))
24     #password_confirm = db.Column(db.StringField)
25     posts = db.relationship('BlogPost',backref='author',lazy=True)
26
27     def __init__(self,email,username,password):
28         self.email= email
29         self.username = username
30         #hashing the password when the user creates it ( constructor password)
31         self.password_hash = generate_password_hash(password)
32
33     def check_password(self,password):
34         return check_password_hash(self.password_hash,password)
35
36     def __repr__(self):
37         return f"Username {self.username}"
38
```

Figure 13 - Fichier *models.py* situé dans le dossier *altariscompanyblog*

La figure 9 montre le début du fichier *models.py*, les lignes après la ligne 37 définissent le modèle du *BlogPost*.

- Les lignes 4, 5, 6 et 7 correspondent aux imports des librairies ou classes nécessaires à ce fichier. On peut constater que j'importe la class *db* depuis mon dossier parent car je vais utiliser des méthodes provenant de cet objet. En effet, dans le fichier *__init__.py* localisé dans ce répertoire, je crée une instance de *db* en utilisant l'extension *SQLAlchemy*. C'est ce que va me permettre de créer mes colonnes de table plus bas.
- De ligne 14 à la ligne 25 je mets en place ma table *User*. Je donne un nom à ma table afin d'y accéder plus tard en ligne 15. Je paramètre par la suite chacune des colonnes en pensant à bien mettre en place une clé primaire sous forme d'identifiant (ligne 18), ainsi qu'une relation avec la table de *BlogPost* grâce à la méthode *relationship()* en ligne 25. Flask me permet de configurer des tables de manière aisée et avec peu de lignes de code.
- Les lignes suivantes démontrent la création d'un constructeur de ma table en tant qu'objet, d'une méthode de vérification de mot de passe et d'une représentation de mon objet *User*.

Une fois mon fichier terminé, je dois effectuer une migration¹⁷ : Flask et SQLAlchemy doivent se synchroniser. Afin de mettre à jour mon système de base de données, je dois taper 4 commandes en terminal me permettant d'indiquer à mon application Flask que je vais créer des tables en base et également en faire des instances. C'est à ce moment précis que le fichier *data.sqlite* se générera.

Mon fichier des modèles a un rôle spécifique qui est celui de garantir un accès réservé vers la base de données, et ce, en se situant distinctement éloigné de mes vues (répertoire Templates). Ce sont des fichiers de traitements logiques *.py* qui auront l'accès au fichier *models.py*.

5.2.4 Produire du logiciel en équipe

J'ai été amené à travailler en équipe à plusieurs reprises au cours de l'année U'DEV. Un exemple que je souhaite mettre en avant est une API¹⁸ réalisée en pair-programming¹⁹ avec un collègue sur un projet à l'origine débuté à l'EPSE. Cette API a été codée en utilisant Java Spring Boot et Spring Data.

L'objectif fut à la fois de développer une API fonctionnelle, mais également de pouvoir monter en compétence dans le travail d'équipe et plus particulièrement dans l'agilité. Nous avons souhaité voir notre API comme un « produit » modifiable à tout moment. Nous avons aussi établi que nous pouvions tous les deux mettre à jour les spécifications détaillées et apporter des suggestions de modification à tout moment. J'ai donc développé une API en Java Spring en équipe, sous forme d'approche incrémentale. Cette activité m'a permis d'apprendre à écrire du code de manière pointilleuse car je savais que mon collègue aurait besoin de relire mon code à un moment donné et versa.

Concevoir une API m'a apporté un atout en plus : celui de concevoir un logiciel disponible d'utilisation pour autrui.

5.2.5 Concevoir des éléments logiciels opérationnels, génériques et réutilisables/partageables

Dans cette partie, je souhaite me focaliser sur l'API mentionnée dans la section du travail d'équipe. Une API est un logiciel intermédiaire permettant à des applications de communiquer entre-elles. Nous pouvons voir une API comme un livreur, nous fournissant ce que nous (le client) souhaitons obtenir. Dans ce cas précis, ce que souhaite obtenir le client en général; et par client j'entends l'utilisateur qui via son navigateur web lance sa recherche ou son clique sur un bouton; c'est une ressource. Une ressource est en réalité une URI²⁰ car c'est elle qui contient les données convoitées par le navigateur. C'est une chaîne de caractères identifiant tout type de ressource sur un réseau.

La communication envers l'API réalisée doit se faire par le biais de requêtes HTTP. L'objectif de la tâche à accomplir est d'envoyer une requête vers l'API afin que celle-ci nous renvoie une réponse. Le principe de question/réponse est fondamental lorsque nous travaillons avec des APIs.

J'ai développé en binôme une API capable de gérer du matériel réseau pour un ou x clients donnés. Ce produit est partageable et réutilisable pour toute personne souhaitant par exemple afficher une liste de matériels pour telle ou telle entreprise. Un des premiers objectifs était du pouvoir requêter

¹⁷ Flask Migrate est l'extension permettant de gérer les migrations

¹⁸ Application Programming Interface.

¹⁹ Attention à bien différencier le peer-programming qui représente un binôme travaillant sur le même processus alors que le pair-programming est une pratique « agile » sur ce même processus.

²⁰ Uniform Resource Identifier. On emploie plus communément le terme URL.

cette API depuis une application Android, tout comme depuis une application web. L'API, à réception de chaque requête, doit aller chercher les données demandées dans une base de données créée à cet effet.

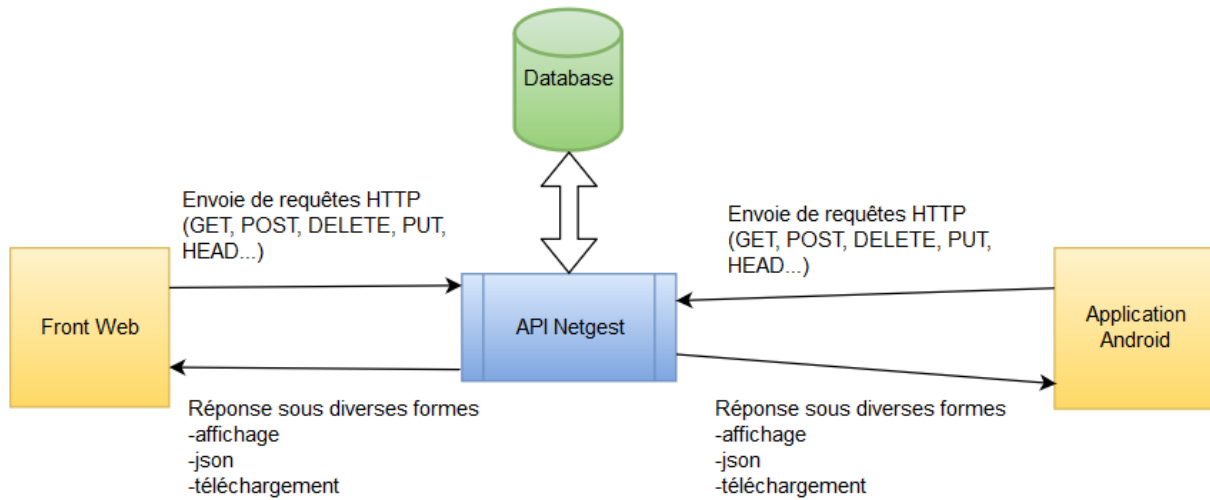


Figure 14 - Schéma relationnel des différentes entités du projet en équipe Netgest

Pour communiquer, j'utilise le format d'échange de données le plus utilisé au monde, le JSON²¹. Le logiciel me renvoie les données requêtées sous forme d'objets JavaScript. L'outil d'émission de requête HTTP utilisé pour ce travail est Postman. Il nous offre une interface avantageuse, nous permettant de lancer tout type de requête et de visualiser en direct le résultat au format JSON ou en données bruts dans un champ inférieur de l'interface.

L'API sous le Framework Java Spring se présente comme le montre l'image suivante :

²¹ JavaScript Object Notation.

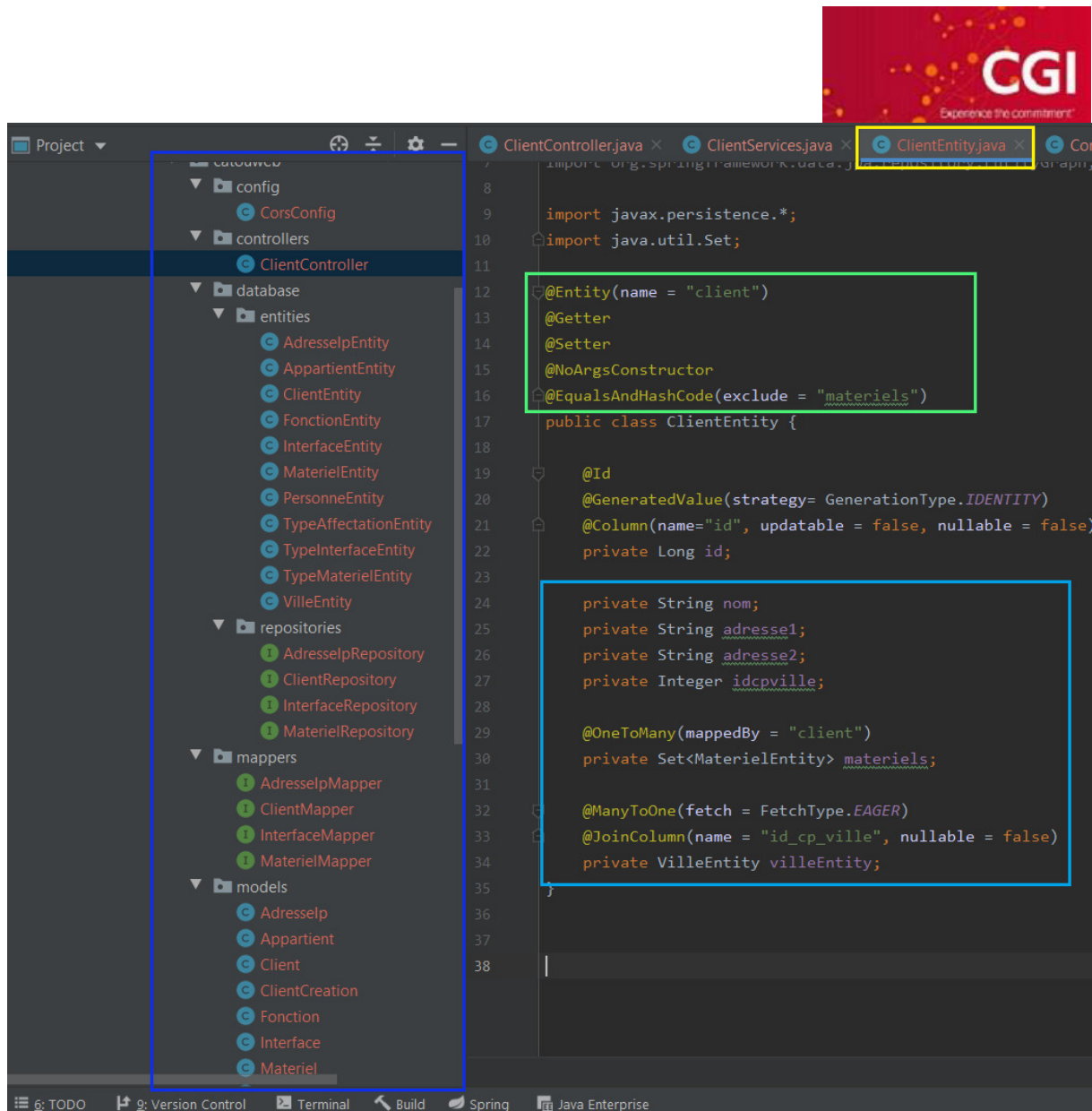


Figure 15- Architecture d'un projet Spring et classe `ClientEntity` provenant du répertoire `Entity`

L'architecture de Spring se décompose en plusieurs répertoire afin de séparer l'application en plusieurs couches.

Je mets l'accent sur une classe `ClientEntity`, provenant du répertoire `Entity`. Ce dernier caractérise la définition des objets de type `Client` pour notre API.

Je crée par la suite un controller `ClientController` qui va se charger d'appeler les méthodes HTTP :

```

ClientController.java x ClientServices.java x ClientEntity.java x ClientRepository.java x Client.java x MaterielMapperImpl.java x M.
13 @RestController
14 @RequestMapping("/clients")
15 public class ClientController {
16
17     @Resource
18     private ClientServices clientServices;
19
20     @GetMapping
21     public Set<Client> findAll() { return clientServices.findAll(); }
22
23
24     @GetMapping("/{clientId}")
25     public Optional<Client> findById(@PathVariable("clientId") Long clientId) {
26         return clientServices.findById(clientId);
27     }
28
29
30     @PostMapping
31     public Client create(@RequestBody ClientCreation clientCreation) {
32         return clientServices.create(clientCreation);
33     }
34
35     @GetMapping("/{clientId}/materiels")
36     public Set<Materiel> getClientMaterielList(@PathVariable("clientId") int clientId){
37         return clientServices.getClientMaterielList(clientId);
38     }
39
40
41     @PostMapping("/{clientId}/materiels")
42     public Materiel addMaterial(@PathVariable("clientId") Long clientId, @RequestBody Materiel materiel) {
43         return clientServices.addMaterial(clientId, materiel);
44     }
45
46     @DeleteMapping("/{clientId}/materiels/{materielId}")
47     public void deleteMaterial(@PathVariable("clientId") int clientId, @PathVariable("materielId") int materielId) {
48         clientServices.deleteMaterial(clientId, materielId);
49     }
50 }

```

Figure 16 - Un controller Client du projet Netgest en Java Spring

Le controller se charge donc de créer les méthodes responsables de requêtes HTTP.

- En ligne 15 je crée une ressource privée de type *ClientServices* provenant de ma classe *ClientServices* afin de pouvoir l'utiliser. J'effectue un traitement sur un service.
- Dans les lignes suivantes se trouvent les différentes méthodes appliquées à un client. Celles-ci doivent être vérifiées en utilisant Postman.

L'utilisation de Postman gère l'envoi de questions HTTP :

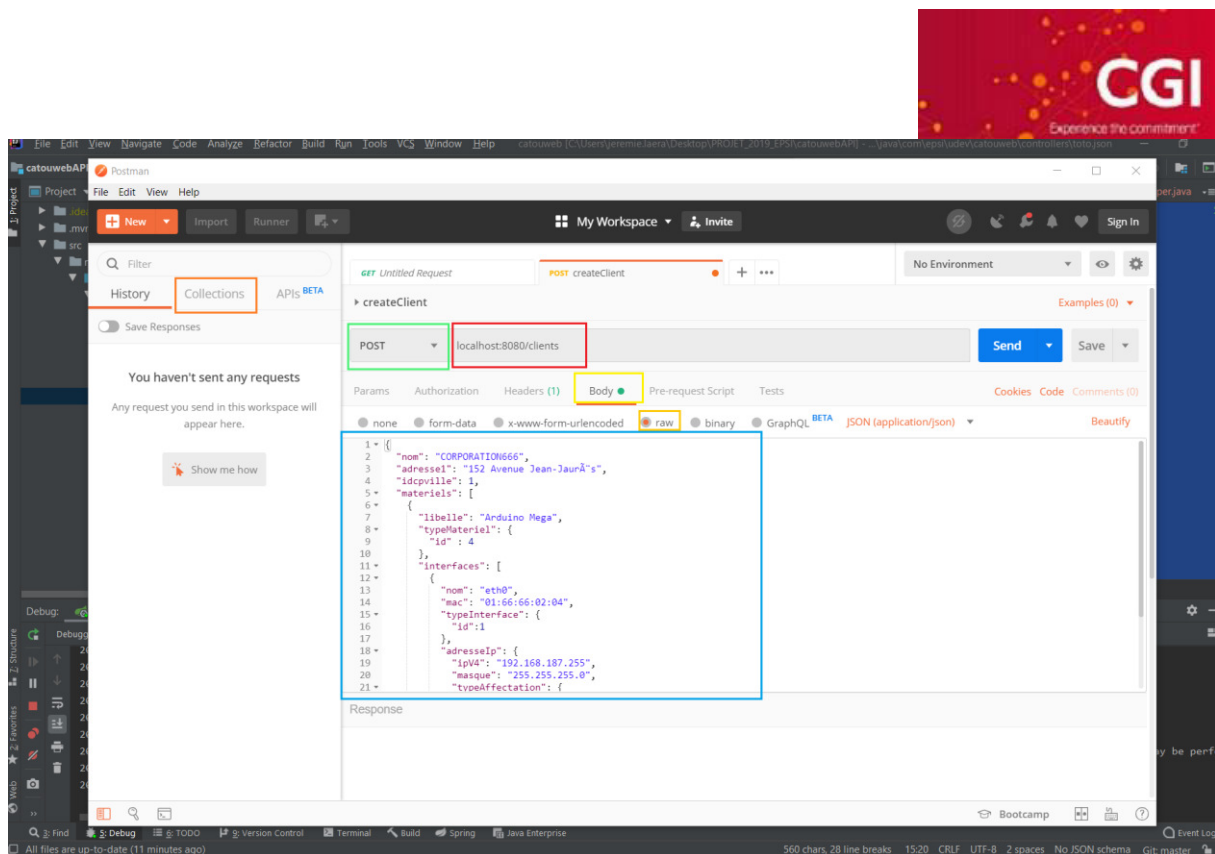


Figure 17 - Requête de création de client passée sur Postman et résultat en données bruts

La figure 16 est une capture d'écran réalisée après le test d'envoi d'une requête de type POST, à la ressource suivante : `127.0.0.1 :8080/clients`. Cette ressource se décompose ainsi :

- `127.0.0.1` correspond à l'adresse de ma machine locale donc à « localhost ».
- Une requête de type POST a été émise (encadré vert), ce qui signifie que j'ai souhaité envoyer des données à un serveur (ici le serveur embarqué dans Java Spring) et que j'ai attendu sa réponse. Le port occupé par le serveur est le 8080 par défaut.
- `/clients` en l'endroit exact où je souhaite envoyer mes données. C'est un endpoint²² qui me sert de chemin dans le but d'envoyer ma requête à un emplacement spécifique.
- La réponse du serveur apparaît dans un « Body » accompagné d'un code de statut HTTP non-visible sur l'image mais égale à 200 dans ce cas de figure. De manière brève, 200 signifie que la requête s'est bien déroulée et que le serveur a accepté les données envoyées.

Les APIs et leur fonctionnement démontrent que pour toute application informatique dynamique la relation client/serveur est essentielle. Les deux entités se renvoient la balle via des questions/réponses grâce au(x) protocole(s) HTTP. Il en va de même pour les applications web et les technologies telles que PHP ou tout autre langage axé côté serveur. Le navigateur est le client et il va requêter un serveur pour obtenir des ressources et les afficher à l'utilisateur qui les demande.

5.2.6 Compétences transverses

En plus des compétences techniques et fonctionnelles, la formation U'DEV enseigne des compétences dites transverses. Ajoutées à celles-ci, se trouvent des méthodes de travail enseignées au cours du cursus.

²² Point final, c'est-à-dire les paramètres passés en fin d'une URI.

5.2.6.1 Méthodes Agiles

C'est notamment le cas de l'apprentissage des méthodes agiles, et plus spécifiquement de l'apprentissage de Scrum²³. Nous avons appris à l'école EPSI à comprendre la philosophie des méthodes agiles et comment piloter un projet en agilité. Nous avons également appris à nous mettre dans la peau des différents agents principaux de la méthodologie Scrum, comme le product owner, le scrum master et l'équipe même sans oublier les stakeholders²⁴. Après de multiples simulations de sprints, nous avons appris à effectuer des rétrospectives et à apporter de nouvelles évolutions sur nos produits. La méthode SCRUM est celle que nous avons le plus étudiée. En méthodes agiles et comme ma rédaction a pu le montrer au fil de cet écrit, le logiciel est vu comme un « produit ».

Néanmoins, nous avons effleuré d'autres méthodologies et philosophies de gestion et pilotage de projets informatiques comme l'Extreme Programming (XP) ou bien la méthode Kanban issue de l'industrie automobile japonaise. Cette dernière a pour vocation de se focaliser sur l'amélioration continue d'un produit afin de minimiser les stocks et le gaspillage. Cette méthodologie se rapproche de l'agilité.

5.2.6.1 Anglais

J'ai également reçu des cours d'anglais ainsi que des épreuves de TOEIC blancs afin de tester mes capacités à écrire et communiquer en anglais. Etant bilingue anglais, ces enseignements ont pu m'apporter des confirmations sur mes compétences actuelles dans cette langue. Les enseignements reçus à l'EPSI sont de la grammaire, conjugaison, compréhension orale et écrite ainsi que les tests de TOEIC blancs.

5.2.6.2 Outil de travail d'équipe

J'ai appris à concevoir et mettre en place des planificateurs de tâches notamment à l'EPSI, en ayant recours à la solution en ligne Trello. Il est facile d'utilisation et permet à tous les membres d'un groupe de travail actifs de se tenir informé de la progression de chacun pour une tâche donnée. Je sépare généralement mes listes de tâches sous la forme de trois voire quatre colonnes : « To-do », « Doing », « Done » et « Inc ».

²³ Origine de la traduction anglaise, ce terme a le sens de mêlée car en Scrum se produisent des « mêlées quotidiennes ».

²⁴ Intervenant extérieur au projet.

6. Conclusion

L'année U'DEV ainsi que l'alternance à CGI m'ont apporté différentes compétences nécessaires à la bonne compréhension du monde de l'informatique, plus particulièrement dans le développement de logiciels et applications.

L'approche du développement sous divers aspects a été effectuée. La conception et le développement de composants informatiques est un métier au sens large. Certains développeurs sont spécialisés dans les applications web, d'autres s'exécutent dans des applications embarquées, et d'autres comme moi travaillent sur des gros systèmes. La forte demande actuelle donne l'opportunité à chacun de pouvoir se développer dans un sous domaine spécifique mais la porte à une réorientation n'est jamais close.

Il me semble qu'un concepteur doit aussi être à l'aise avec les calculs et la manipulation des chiffres et autres types de données : je parle de computation. Bien entendu, il ne s'agit pas d'être un mordu de mathématiques et de mécanique, cependant, je trouve que l'informatique permet de réconcilier tout attitude néfaste perçue ou vécue pour les mathématiques au cours de l'enseignement secondaire. Un ordinateur est avant tout une machine qui ne connaît que les 0 et 1, et qui effectue des opérations arithmétiques. Comprendre comment un langage fonctionne et comment nous arrivons à traduire les données sous format compris par l'être humain est un facteur avantageux pour devenir un informaticien aguerri mais pas nécessairement indispensable. C'est pour cette raison qu'un objectif personnel que je me suis fixé, serait d'apprendre et de comprendre de l'assembleur afin d'effectuer des opérations complexes.

Cette année a su m'ouvrir des portes sur des domaines qui peuvent être développés à plus long terme comme le Big Data, l'introduction aux Intelligences Artificielles et ou encore l'étude de gestion de projets informatiques. Une progression demeure toujours envisageable dans les technologies du numérique qui ne cesseront de croître et d'évoluer, demandant une adaptation continuelle.

7. Références

Robert Wingate. (2018) - COBOL – Basic Training using VSAM, IMS and DB2 — 24 Juillet 2018

Mike Murach - Mike Murach & Associates Inc. (2004) - Murach's Mainframe COBOL — 16 Août 2004

<https://www.cgi.com/fr/media/brochure/cgi-histoire> - Consulté le 03 décembre 2018 – Histoire et évolution de CGI.

<https://en.wikipedia.org/wiki/COBOL> - Consulté le 02 Janvier 2019

<https://www.techopedia.com/> - Consulté le 10 Février 2019

<https://www.techopedia.com/the-history-of-the-c-programming-language/2/32996> - Consulté le 25 Avril 2019 – Article sur les fondements et débuts du langage C.

<https://www.cgi.fr/fr-fr/qui-sommes-nous/cgi-france> - Consulté le 02 Octobre 2019

<https://www.ibm.com> – Consulté pour la première fois le 03 décembre 2018

<http://www-igm.univ-mlv.fr/~dr/XPOSE2000/mvs/> - Consulté pour la première fois le 04 décembre 2019

<https://www.ib-formation.fr/catalogue/nbs-details/catref/universib-formations-informatiques-ibm-system-z/ref/sr831/ibm-z-os-competences-fondamentales> - Consulté le 29 Septembre 2019

Robert C Martin – Clean Code

Robert C Martin – Clean Architecture: A Craftsman's Guide to Software Structure and Design – 20 Septembre 2017

Ian Goodfellow, Yoshua Bengio, Aaron Courville - Deep Learning — MIT Press – 18 Novembre 2016

8. Annexes

Tableau des compétences du référentiel IPI : tableau énumérant les blocs de compétences à valider.

Bloc de compétences : Qualité et sécurisation du code réalisé (reporté en page suivante).

Compétences ou capacités qui seront évaluées	Critères d'évaluation	Exemples d'activités et tâches	Activités pratiquées	Origine de l'acquisition	Preuves apportées & ref. annexe
- Formaliser, identifier les résultats attendus.	La liste de contrôle des attendus fonctionnels est paraphée.	Etude de l'existant. Rédaction du cahier des spécifications fonctionnelles.	Etude d'un DCG et rédaction de DCOD.	E	
- Respecter des contraintes.	Un plan d'assurance qualité est observé.		Ecrire un programme conforme au Plan Qualité de Code	E	
- Respecter les recommandations qualité de la norme en vigueur pour l'architecture des logiciels.	L'application est organisée en couches indépendantes.	Conception/architecture d'applications logicielles. Conceptions de services métiers.	Réalisation d'application bénéficiant d'une architecture N'Tiers	F/B	
- Anticiper les évolutions.	Les règles métier sont encapsulées dans des services logiciels.	Conception de services d'accès aux données.	Réalisation d'une couche de Modèle avec Flask et Python	F/B	
- Qualifier les risques	L'accès aux données est réalisé par des services logiciels indépendants du mode de stockage.				
- Respecter une norme de présentation des écrans et documents de sortie.	L'exécution de l'application est répartie entre un nombre d'ordinateurs adapté au contexte.	Détermination du nombre de tiers de l'application. Réalisation des maquettes de sorties interactives. Réalisation des maquettes de sortie imprimée.	Réalisation d'une application de publication de postes. //	F/B F/B	
	Un formulaire		Prototypage d'application web.	F/B	

	<p>d'estimation des risques est rempli.</p> <p>Une norme de présentation des données est respectée. Les interfaces Homme/Machine sont validées.</p>	<p>Estimation, qualification des risques sécurité.</p> <p>Réalisation d'une interface Homme/Machine.</p>	<p>Qualification des risques estimée.</p> <p>Création de pages HTML, CSS et JavaScript + Bootstrap.</p>	<p>E</p> <p>F</p>	
<p>Concevoir des programmes avec une orientation objets.</p> <p>Garantir un accès sécurisé aux données.</p> <p>Livrer le logiciel déverminé</p>	<p>Une programmation orientée objet est utilisée.</p> <p>Le taux de réutilisation du code utile est > 80%. Des gabarits sont utilisés. Une charte de nommage est utilisée. Le taux de documentation interne du code est > 8% et < 15%.</p> <p>Les anomalies d'accès aux données ne génèrent pas d'interruption de l'exécution et son</p>	<p>Programmation de logiciels.</p> <p>Programmation de l'accès aux données de l'entreprise.</p> <p>Tests Unitaires. Préparation des jeux de tests.</p>	<p>Utilisation de Python avec Flask. Création d'objets Python.</p> <p>Création et modification d'une couche d'accès aux données.</p> <p>Réalisation de tests unitaires et débogage d'applications.</p>	<p>B</p> <p>E/F</p> <p>E</p>	

Livrer le logiciel conforme aux attentes.	répertoriées.				
	Des outils de contrôle automatique du code sont utilisés. Aucun défaut visible ne persiste. Les contraintes spécifiques au projet sont respectées. Un manuel d'assurance qualité est respecté. Une méthode de recettage est utilisée. L'étape du projet est validée.	Contrôle de l'existence d'anomalies. Recettage du logiciel. Validation d'une étape du projet.	Vérification de cas de tests unitaires. Confirmation et livraison de fiches de tests unitaires. //	E E //	
Clôturer une mission.	Le PV de réception du logiciel est validé.	Mise en exploitation.	Mise en exploitation de documents livrables.	E	

Bloc de compétences : Audit, conception, méthode de projet

Compétences ou capacités qui seront évaluées	Critères d'évaluation	Exemple d'activités et tâches	Activités pratiquées	Origine de l'acquisition	Preuves apportées & ref. annexes
Formaliser des processus	La procédure du service utilisateur est formalisée et validée.	Etude de l'existant. Identification des procédures en place.			
	La procédure du service utilisateur est conforme aux règles du système de management des services de l'entreprise.	Contrôle de la conformité des procédures utilisées avec la gouvernance de l'entreprise.			
	La circulation du document résultat du traitement prévu est matérialisée dans un diagramme de workflow.	Recensement des documents utilisés, identification de leur circulation et des acteurs concernés.			
Formaliser les règles de gestion et d'organisation des données de l'entreprise.	La proposition de reconstruction de la procédure est validée.	Reconfiguration de procédure.			
	La base de données est modélisée.	Conception d'une base de données.	Conception d'une base de données MySQL.	F	
Une méthode de conception par objets est utilisée.	Concevoir des éléments logiciels réutilisables.	Conception de l'architecture applicative.	Conception d'une API avec la programmation orientée objet.	F/B	
Une méthode AGILE est	Produire du logiciel en	Programmation en équipe.	Travail avec la méthode Scrum	F/B	

utilisée.	équipe.	Ecriture de code.			
Absence de signaux d'alertes au point de contrôle du projet.	Remonter les alertes au(x) décideur(s).	Coordination de l'avancement.	Avertissement et contact avec les collaborateurs.	E/F/B	
Les étapes du projet sont planifiées.	Estimer des délais.	Planification des tâches du projet.	Estimation des délais et reste à faire (RAF).	E/F/B	
Le projet est conforme au schéma directeur de l'entreprise et respecte les principes d'urbanisation du SI.	Concevoir une solution logicielle.	Conception de la solution logicielle.	Ecriture de DCODs et conception de programmes mainframe.	E	
Les spécifications fonctionnelles produites respectent le cahier des charges fourni.					
L'impact de modification est acceptable.	Anticiper des répercussions.		Préparation des algorithmes et cas de tests de non-régression.	E	

Bloc de compétences : Réalisation d'applications logicielles

Compétences ou capacités qui seront évaluées	Critères d'évaluation	Exemples d'activités et tâches	Activités pratiquées	Origine de l'acquisition	Preuves apportées & ref. annexe
Encapsuler des solutions logicielles spécifiques dans des services logiciels génériques.	Le service d'accès aux données est opérationnel.	Programmation. Investigation documentaires fonctionnelles ou techniques complémentaires.	Réalisation de programmes mainframe. Ecriture/lecture de DCODs.	E E	
Produire du logiciel générique réutilisable.	Des services logiciels internes sont réutilisables.	Transcription des spécifications fonctionnelles en algorithmes.	Ecriture de DCODs et d'algorithmes fondés sur des DCGs.	E	
Produire du logiciel partageable.	Des services logiciels sont partageables en local.	Transcription des algorithmes en code source. Compilation, déverminage du code source.	Ecritures de programmes batchs et SA basés sur des DCODs. Compilation et débogage de programmes batchs et SA.	E E	
Intégrer des éléments logiciels hétérogènes et produire des exécutables livrables.	Le logiciel est livrable, prêt pour la mise en production.	Agglomération des différents éléments logiciels en unités de traitement, réalisation de tests unitaires.	Réalisation de tests unitaires, et tests d'assemblage pour des SA.	E	
Modifier un algorithme sans générer de dysfonctionnements.	La modification n'entraîne pas de régression fonctionnelle.				
Contrôler des délais.	Le compte-rendu d'activités est renseigné, les écarts sont constatés.	Mise à jour du planning de réalisation.	Actualisation du planning et imputation sur les charges prévues pour les demandes.	E	

Bloc de compétences : Communiquer avec les acteurs du projet

Compétences ou capacités qui seront évaluées	Critères d'évaluation	Exemple d'activités et tâches	Activités pratiquées	Origine de l'acquisition	Preuves apportées & ref. annexes
User d'une communication professionnelle tant en français qu'en anglais. Interagir efficacement dans un environnement de travail collaboratif.	Le compte-rendu de la réunion est validé.	Elaboration et rédaction de documents techniques, commerciaux ou internes à destination, des utilisateurs, des clients ou des collaborateurs...	Ecriture de documents et code en anglais.	E	
	Le score du TOEIC est > 749.		TOEIC Blanc avec un score s'élevant à 910.	F	
	Le document collectant l'expression des besoins des utilisateurs est validé.	Rédaction des spécifications fonctionnelles de la solution informatique.	Rédaction et modification d'un DCG.	E	
	L'aide du logiciel est rédigée. La documentation du livrable est diffusée.	Ecriture des interfaces Homme/Machine Relations avec les clients. Animation de réunions de travail et interviews d'utilisateurs.	Participation à des réunions de co-pilotage. Daily Meeting.	E	
	La présentation est appréciée. Les utilisateurs sont opérationnels, le	Démonstration, recettage de livrables. Formation des utilisateurs au logiciels.	Création de package sous ENDEAVOR et eDMP et mise à disposition.	E	

	transfert des nouvelles compétences est validé.				
--	---	--	--	--	--

Bloc de compétences : Adapter l'environnement d'exécution, échanger des données entre logiciels

Compétences ou capacités qui seront évaluées	Critères d'évaluation	Exemples d'activités et tâches	Activités pratiquées	Origine de l'acquisition	Preuves apportées & ref. annexe
Réaliser des échanges de données informatisés (EDI).	Les données sont consolidées.	Réalisation d'un procédé d'échange de données informatisées.	Envois de fichiers via programmes batchs vers d'autres applications demandeuses.	E	
Automatiser des traitements.	La base de données tierce est accédée.	Rétro-documentation de logiciels et de bases de données.	Création et modélisation de schémas VISIO.	E	
	L'interface d'échange de données est opérationnelle.	Consolidation, agrégation de données. Programmation de l'interface d'échange de données.	Automatisation de traitements avec eDMP. Création de packages contenant des scripts automatisés.	E E	
		Réalisation d'un environnement de tests. Création, configuration de machine virtuelles. Installations, configurations de serveurs d'applications, web et base de données.	Utilisation des options de programmation pour les JCLs. Création de machines virtuelles avec Virtual Box. Dockerisation d'applications Python. Création de JCLs tournant en environnement de test sous émulateur TSO.	E B E	
Programmer des scripts systèmes.	L'environnement de tests est opérationnel.	Ecriture de scripts systèmes pour adapter l'environnement d'exécution.	Création des scripts JCL adaptables pour tout environnement (tests unitaires, recette, pré-production et production).	E	



Curriculum Vitae – Novembre 2019

Jérémie LAERA

20/01/1992

Master Etudes des Mondes Anglophones

5 Rue Salvador Allende

Concepteur/développeur mainframe

Apt 178

33400 Talence

Jeremie.laera@live.fr

Expériences professionnelles

2018 - 2019 : Développeur et concepteur d'applications mainframe - CGI - 33187 Le Haillan

2017 - 2018 : Gestionnaire et livreur de commandes de plats préparés - Uber Eats - 33000 Bordeaux

2017 - (Juillet - Décembre) : Agent polyvalent au sein de Ministère de l'Intérieur - Préfecture de la Gironde - 33000 Bordeaux

2016 - 2017 : Assistance en animation - Night Music Events - 86000 Poitiers

2015 (temps-partiel) : Assistant en dépannage informatique et webmastering - Office 42 - 86000 - Poitiers

2014 (Eté) : Préparateur / livreur de commandes - E.Leclerc - 86100 - Châtelleraut

2011- 2013 : Agent de collecte d'ordures ménagères - CGT - 86100 - Châtelleraut

Formations

2018 - 2019 : U'DEV2 - EPSI - 33000 - Bordeaux

2015 - 2016 : Master Recherche Etudes des Mondes Anglophones - Université Bordeaux Montaigne - Pessac 33600

2013 - 2014 : Licence Langues Littératures et Civilisations Etrangères (LLCE) Anglais mineure Espagnol - UFR Lettres et Langues - 86000 - Poitiers

2012 - 2013 : DEUG Langues Littératures et Civilisations Etrangères (LLCE) Anglais mineure Espagnol - UFR Lettres et Langues - 86000 - Poitiers

2009 - 2010 : Baccalauréat Economique et Social - Lycée Marcelin Berthelot - 86100 Châtelleraut

Compétences et Loisirs

Sport : Football, Athlétisme, Fitness, Badminton

Bureautique : Prezi, Suite Office

Conception et programmation : mainframe, web et API (Java, Python, PHP), IA (Python), C, Kotlin

Langues : Français, Anglais, Italien

Fiche d'évaluation de mon tuteur Emmanuel Lafitte



CERTIFICATION PROFESSIONNELLE EVALUATION ENTREPRISE

STAGIAIRE

Nom : **LAERA**
Prénom : **JEREMIE**

Certification visée : **RNCP 30714**
Conception, Développement
d'Applications Numériques
«tuteur_Civilite».

Afin de parfaire la validation de la certification professionnelle de votre stagiaire, nous vous remercions de remplir exhaustivement cet **original** et nous le retourner dans les *meilleurs délais*. Ce document sera examiné par le jury en vue de l'attribution de la certification professionnelle de votre stagiaire.

ENTREPRISE

Nom de la société : **CGI**
Tuteur(trice) : **LAFITTE EMMANUEL**
Courriel : **emmanuel.lafitte@cgi.com**

L'intégration dans l'équipe de travail est-elle satisfaisante ?	Oui / Non
Le comportement en situation professionnelle est-il adapté à la culture de l'entreprise ?	Oui / Non
Le comportement relationnel est-il adapté au métier visé ?	Oui / Non
La capacité de travail fournie correspond-elle au niveau d'un professionnel du métier ?	Oui / Non
Le niveau de responsabilités attendu est-il satisfait ?	Oui / Non
Des questions sont-elles posées à bon escient ?	Oui / Non
Des difficultés en relation avec le niveau de responsabilités sont-elles surmontées ?	Oui / Non
Le niveau d'obligation de réserve demandé est-il pris en compte ?	Oui / Non
Les activités effectuées concourent-elles à la couverture du champ de compétences ?	Oui / Non
Le métier associé à la certification visée est-il compris ?	Oui / Non
Le stagiaire est-il apte à occuper la fonction visée même comme débutant ?	Oui / Non

Appréciation générale suite au stage en entreprise :

Suite à une première période de formation sur l'environnement Nainfrance, Jérémie s'est bien intégré dans l'équipe du CDS REF de LBP. Après une période nécessaire d'accompagnement, Jérémie est monté en compétence sur les tâches de Réalisation et Tests Unitaires, ce qui lui a permis de se voir confier la réalisation de Dossiers de Conception Détaillée. Tout au long de son stage, Jérémie a su se montrer ponctuel et professionnel.

Fait à **Le Maillet**, le **12/11/13**

Signature du tuteur
et cachet de l'entreprise



Signature du stagiaire



Association ADIP regie
par la loi du 1^{er} juillet 1901
Siret : 409 801 677 00017 Code
APE : 85 59 A
N° organisme : 11 75 27 97 775

Centre de formation
44 bis, quai de Jemmapes 75010
Paris
Siège social
12, rue Alexandre Parodi 75010 Paris
S.A.S. au capital de 137 913 933 € 11 80 97 35 00
RCS Nanterre B 701 042 755
m.lafitte@groupe-igs.fr
www.ipi-ecoles.com



© Institut de Poly-informatique (2011)