Rapport de stage

Développement d’un langage de programmation en C.

7 juil - 5 déc 2025

Liam Lequet

Société d’accueil : kmx.io

Établissement : Epitech Paris

Année: 2025/2026

Mention légale:

L'auteur, Liam Lequet, se réserve tous les droits sur le présent document.
Tout usage externe de ce dernier à des fin autres que son évaluation dans le cadre du cursus académique de l’auteur est donc interdit sans le consentement écrit et préalable de l’auteur.

Remerciements

Remerciements :

Premièrement, je tiens à remercier **M. Thomas de Grivel**, à la tête de **kmx.io**.
En tant que maître de stage, il m'a transmis son expertise, sa rigueur
et sa vision dans le domaine de l'ingénierie logicielle, ce qui m'a permis de mûrir, de monter en compétences et d'affiner mes objectifs professionnels.

J'adresse également ma gratitude envers mes deux co-stagiaires, ces derniers m'ont été d'un soutien inestimable durant ce stage.

Je remercie sincèrement les membres de l'équipe pédagogique d'Epitech pour leurs conseils et leur accompagnement durant ces cinq mois, sans lesquels ce stage n'aurait pas pu avoir lieu.

Enfin je témoigne ma sollicitude envers mes parents pour leur inconditionnel soutien moral et financier.

Table des matières :

Glossaire

Glossaire :

Terme	Définition
S.A.S.U	Société par actions simplifiée unipersonnelle, société commerciale pouvant exercer tout type d'activité, à l'exception de certains secteurs réglementés.
P.M.E	Petites et moyennes entreprises, société dont l'effectif est inférieur à 250 personnes et dont le chiffre d'affaires annuel n'excède pas 50 millions d'euros.
Langage C	Le langage C est un langage de programmation généraliste inventé au début des années 1970. Ce langage est encore largement utilisé et en a inspiré beaucoup d’autres.
Administration système	Gestion et maintenance des systèmes informatiques et réseaux d'une organisation (ex : configurer des serveurs, gérer les droits utilisateurs).
DevOps	Le DevOps est un domaine de l’informatique faisant le lien avec d’autres domaines autrefois séparés à l’aide d’outils spécialisés.
Scrum Master	Le Scrum Master est la personne responsable de l’organisation d’une équipe employant une méthodologie scrum, c’est elle qui définit les objectifs de cette dernière et veille à leur accomplissement.
Production	La production est un terme désignant la phase finale du développement. C’est là que tout ajout fini et testé réside pour être intégré à la prochaine version d’un logiciel.
Git	Outil traquant les changements opérés dans un environnement de développement et permettant leur gestion.

Répertoire ou Référentiel	Un répertoire Git est un environnement de développement hébergé sur une machine et permettant l'usage de Git pour y ajouter et modifier du code.
Bug	Un bug (ou bogue en français), est un comportement imprévu de la part d'un logiciel, ils sont généralement nocifs et donc indésirables.
Code review	Une code review est un entretien visant à passer en revue du code produit par un développeur afin d'en assurer la qualité et d'améliorer certains points pour l'avenir.
Système d'exploitation	Un système d'exploitation est un logiciel permettant de faire usage des composants physiques d'un ordinateur.
Architecture processeur	Ce terme désigne l'agencement et les spécifications physiques d'un processeur.
Paradigme de programmation	Approche de la programmation informatique et méthode d'élaboration de solutions avec des outils en accord avec cette dernière.
Prog. procédurale	La programmation procédurale est un paradigme qui se fonde sur le concept d'utilisation explicite, toute instruction est écrite sans surcouche médiane.
Prog. Orientée Objet	Paradigme consistant en la définition et l'interaction de d'entités logicielles appelées objets. Un objet représente un concept, une idée ou une entité physique.
Prog. Fonctionnelle	Paradigme de programmation déclaratif qui considère le calcul en tant qu'évaluation de fonctions mathématiques.

Graphe	Un graphe est un ensemble d'objets, appelés sommets ou parfois nœuds liés mutuellement par des arêtes
Nœud	En théorie des graphes, un nœud (ou vertex) est un sommet pouvant être relié à un autre par un chemin direct.
Base de données	Une base de données est un système de tables consultables servant à stocker des données de façon ordonnée.
Fonction	Composant logiciel réutilisable performant une série d'actions par rapport à ce qui lui est donné en entrée et fournit une valeur qui en résulte.
Binaire	Représentation de nombres avec des suites de bits (1 ou 0). Chaque bit représente une puissance de 2 à prendre en compte ou non dans leur addition pour représenter un nombre.
Environnement	L'environnement est un ensemble de variables de références sur laquelle un programme se base pour exécuter des instructions de la bonne manière.
Octet	Un octet (anglais : byte) est un ensemble de 8 bits succesifs.
Hexadécimal	Représentation de nombres avec 16 chiffres (de 0 à 9 puis de A à F), généralement utilisée pour représenter des valeurs binaires de façon plus compacte.
Librairie	Ensemble de fonctions ayant un but commun, prêt à l'emploi et importable dans un programme.

UTF-8	Anglais : Universal Character Set Transformation Format - 8 bits. Encodage de caractères informatiques universel standardisé.
Man in the middle	Cette expression désigne une personne ou un ordinateur se positionnant au milieu d'un échange de données entre deux machines pour les extraire à des fins malveillantes.
Compression	Méthode de réduction de la taille en octets d'un ensemble de données pour faciliter sa transmission.
Terminal	Logiciel permettant à l'utilisateur d'interagir avec une machine via des commandes.
Compilation	Conversion d'instructions lisibles par l'Humain en instructions exécutable par un ordinateur.

Légende :

terme* → Défini dans le Glossaire (Cliquable)
(**voir section** X.X.X) → Redirige vers la page associée
phrase soulignée en bleu → lien Internet (Cliquable)

1. Introduction

1 Introduction :

1.1 Présentation du stage

1.1.1 Contexte

Du **7 juillet** au **5 décembre 2025**, j'ai eu l'immense honneur de me voir accepté au sein de l'entreprise **kmx.io** dans le cadre de son **stage** de 2025.

J'ai été chaleureusement accueilli par son dirigeant **M. Thomas de Grivel,** mon maître de stage. Le présent rapport décrit le déroulé de ce dernier et ses différents aspects.

1.1.2 Présentation de la mission

Au cours de ce stage, j'ai pu contribuer au projet le plus ambitieux de cette structure, ce dernier consiste à améliorer et créer des fonctionnalités pour un nouveau langage de programmation en graphe (voir **section 3.1.1**) nommé "KC3".

L'intégralité des composants ayant permis la création de ce langage est écrite en langage **C** sous le standard **C11**.

Le répertoire utilisé par l'entreprise est public.

1.2 Description de l'entreprise

1.2.1 Contexte fiscal

L'entreprise **kmx.io** est une **S.A.S.U***, considérée comme étant une **P.M.E***.

Son **SIREN** est **914877436** et cette dernière est située dans la ville de **Levallois-Perret**.

La société est détenue et dirigée par **M. Thomas de Grivel** en sa qualité de président et unique actionnaire de cette dernière.

1.2.2 Secteur d'activité

La société **kmx.io** propose un service de location de serveurs dédiés sécurisés.
De plus, elle fournit des solutions techniques et un service de conseil aux entreprises de toute taille. Les différentes missions peuvent consister à de l'optimisation, la résolution de comportements inattendus ou le développement de nouvelles fonctionnalités selon les besoins de l'entreprise commanditaire.

1.2.3 Place dans le marché

Du fait de sa petite taille, l'entreprise n'est connue que d'une poignée d'individus. Cependant, les **services** apportés par la société sont d'une **qualité rare**, ce qui pourrait à l'avenir prédisposer cette dernière à obtenir et conserver une place au sein du marché, et ce de façon durable.

1.2.4 Place dans l'entreprise

En tant que **stagiaire**, mon rôle consistait principalement à mettre mes compétences en **langage C*** au profit d'un nouveau langage de programmation.

Additionnellement et au vu de mes compétences annexes, mon **affectation** s'est **mue** au fil du temps.

Progressivement, je me suis vu confié de plus en plus de tâches en relation avec la maintenance, l'amélioration et la création de solutions périphériques au projet mère.

Ainsi, mon champ d'action s'étendait non seulement au développement en langage C, mais aussi à des domaines tels que l'**administration système***, le **DevOps*** et la gestion de l'environnement de développement.

2. Cadre de travail

2 Cadre de travail

2.1 Organisation

Le stage s'est déroulé autour d'une **méthodologie Scrum**. Chaque **lundi** matin, avait lieu une **réunion** avec l'ensemble des membres du stage afin que le "*Scrum Master**" (ici, le maître de stage) puisse définir les **objectifs hebdomadaire** à remplir.

Durant le reste de la semaine avait également lieu une courte réunion le matin pour définir les objectifs du jour et mettre à jour la liste de tâches à accomplir.

En fin de journée, nous passions en revue les changements apportés pour les intégrer en production*.

2.1.1 Gestion du référentiel Git*

Le **répertoire*** est réparti en plusieurs branches (copies modifiables du répertoire), chaque fonctionnalité du langage **KC3** en possède une dédiée, ce qui permet à **plusieurs personnes** de **travailler** simultanément sur des **fonctionnalités différentes** sans s'entraver mutuellement.

Quand une de ces **fonctionnalités** est **complète** et entièrement testée, il suffit d'**appliquer** les changements de la branche correspondante à la principale.

2.2 Locaux et travail en distanciel

L'entreprise **kmx.io** ne possédant pas de locaux aménagés, le stage s'est déroulé en grande majorité en **distanciel**, avec d'**occasionnelles réunions** en **présentiel**.

Cependant, le manque de locaux n'a pas été une contrainte, avec un régime de travail uniquement distanciel, **travailler à domicile** restait une option.

Néanmoins, le campus Epitech de Paris étant ouvert en permanence aux étudiants, ce dernier facilite grandement l'accès à un cadre de travail plus structuré.

Avec l'**accord** de l'**équipe pédagogique**, la majeure partie du temps passé à travailler en distanciel a été passée sur le campus.

2.3 Journée type

Le déroulé du stage étant relativement cémenté, une journée type ressemblait à ceci :

10h00	11h00	12h00 - 13h00	13h00 - 18h00	18h00
Réunion : Définition des tâches.	Résolution de bugs* ou travail sur fonctionnalités.	Pause déjeuner	Écriture de code en binôme.	Code review* et mise en production

(NB : Ne prend pas en compte certains facteurs externes tel que le temps de déplacement en transport en commun.)

2.4 Environnement technique

En réponse à un **besoin** de **compatibilité** sur tout système d'exploitation*, l'environnement technique se composait d'une multitude d'ordinateurs possédant des systèmes d'exploitation et des architectures processeur* différents.

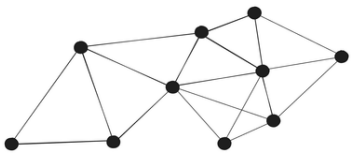
Parmi ces systèmes d'exploitation, les principaux étaient **Linux** (Ubuntu), **BSD** (OpenBSD), **Windows 10/11**, **MacOS**.

Les architectures utilisées étaient **x86-64** (Intel) et **ARM** (Apple, Androïd).

Une telle variété peut sembler déroutante, cependant mon passif d'administrateur système m'a permis de me sentir tout de suite à l'aise avec les différentes machines de l'entreprise.

3. Réalisations techniques

3 Réalisations techniques



3.1 KC3

3.1.1 Programmation orientée graphe

La *Programmation Orientée Graphe* (POG), en anglais *Graph Oriented Programming* (GOP) est un nouveau paradigme de programmation*, au même titre que la Programmation Procédurale*, Orientée Objet* et Fonctionnelle*.

Quelques exemples de langages et leurs paradigmes :

Procédural	Orienté objet	Fonctionnel
C, Go, Pascal...	C++, C#, Java, Python...	Lisp, Haskell...

Seulement **théorisé** vers la fin des années **2010**, ce nouveau paradigme représente les données sous forme de **nœuds*** indépendant, reliés entre eux par des **relations** mutuelles (parent/enfant, voisins etc...).

Ainsi, **exécuter** les instructions données revient à **parcourir** l'entièreté du **graphe*** constitué par le langage et à relier les données le constituant entre elles.

A ce jour, **KC3** est le **seul langage** orienté **graphe** opérationnel **connu** ayant une première révision.

Ce paradigme lui permet notamment de posséder de nombreuses fonctionnalités, dont un **système** de gestion de **base de données*** dynamique et optimisé.

KC3 est **utilisé** à ce jour pour faire fonctionner tous les **sites web** possédés par **kmx.io** en **arrière-plan**.

L'**objectif principal** de la mission était de **développer** les nouvelles **fonctionnalités** du langage.

3.1.2 Syntaxe du langage KC3


```
defmodule Test do
  def one = 1
  def double = fn (x) { x * 2 }
  def add = cfn Tag "tag_add" (Tag, Tag, Result)
end
```

(source : kc3-lang.org)

KC3 est fortement **inspiré** par les langages **C**, **Elixir** et **Common Lisp**.
Tout est centré autour de **modules**, qui sont ensuite convertis en **nœuds**.

Le code ci-dessus crée un nouveau module nommé “Test” et y incorpore des éléments.

Il est aussi **possible** de faire usage de fonctions* **C** à l’intérieur de **KC3**, donnant à l’utilisateur une parfaite maitrise de son programme.

Tous les types de données implémentés en C le sont aussi en KC3.
Ce dernier également vient renforcer la liste avec d’autres types venant d’Elixir et Common Lisp mais aussi ses propres types.

La **syntaxe** de ce langage est **atypique**, voir éloignée de celle de la majorité des langages communs, dont ceux qu’enseigne Epitech. Ceci peut constituer une barrière quand à l’apprentissage du langage.

J’ai pu courtement développer en KC3 dans le but de mieux comprendre le but de l’implémentation de ses fonctionnalités natives crées en C, cela m’a permis de découvrir la théorie relative à la syntaxe des langages de programmation.

3.2 Marshalling de structures de données

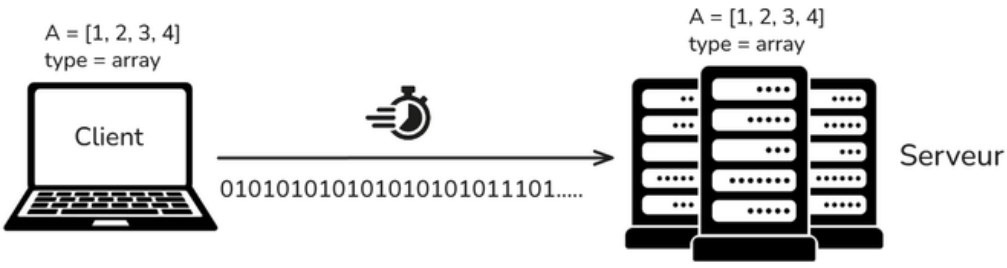
3.2.1 Définition et usage

Le **Marshalling** est un procédé consistant à la **conversion** de données en un **format** plus **adapté** au stockage et/ou la transmission vers d'autres machines, ainsi que leur reconstitution.

Ce dernier comprend une étape de **sérialisation** (voir **section 3.2.4**) et **transforme** des structures de données en un format plus performant, KC3 utilise le format binaire*.

De manière générale, un programme **client** va **encoder** des données à envoyer vers un **serveur** qui les **décodera** et vice-versa.

Attention : Ce procédé n'est aucunement une méthode de sécurisation de transfert de données (voir **section 3.3**)



3.2.2 Application spécifique à KC3

Outre les applications en **stockage** et en **communication** entre machines, le marshalling permet le **chargement** rapide de données en mémoire locale sur une seule machine.

Cette implémentation répond à un besoin d'améliorer la **vitesse** de démarrage de l'**interpréteur** IKC3 (voir **section 3.5**); limitée par celle du **chargement** des variables d'environnement* et de ses autres composants. Elle a permis une **réduction** du temps de démarrage initial moyen de **8000%**, soit une division par quatre-vingt, passant de plusieurs secondes à une poignée de millisecondes.

Il aura fallu deux mois à notre équipe pour implémenter la conversion de l'entièreté des structures de données de KC3, le procédé fut simple mais très long au vu des nombreuses structures de données que possède le langage.

3.2.3 Abstraction de types via tags

Afin de pouvoir **convertir** des données dans un format **binaire**, il faut d'abord élaborer un système permettant d'**identifier** ces **données** et leur type.

Le système interne à KC3 utilise un système de "tags" (en français : étiquettes).

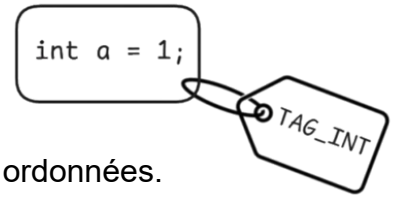
Un "tag" est un **identifiant** attribué à chaque portion de donnée permettant de la **convertir** et de la **reconstituer** depuis et vers le **format binaire** selon son type en faisant usage du bon décodeur, *une bonne analogie serait de coller des étiquettes sur des vêtements afin d'en connaître la couleur exacte*. Cette **méthode** est **courante** dans les procédés de communication Internet.

Ayant réalisé le projet *Corewar* à Epitech, l'utilisation d'identifiants binaire pour exécuter certaines actions n'est pas une barrière.

Que ce soit à l'envoi, à la réception ou au chargement, une **mésinterprétation** de type de donnée est rendue **impossible**.

Ce système existait déjà avant le début du stage, il était prêt à l'emploi. Il s'est avéré indispensable au bon déroulement de ma mission et au développement du marshalling.

3.2.3.1 Représentation en langage C



En langage C, il existe des **enums** (énumérations).
Ces enums sont un ensemble de valeurs entières constantes ordonnées.

Chaque membre d'une énumération est un nombre représenté par un nom.

Dans le code source du langage KC3, chaque type de donnée possède un identifiant qui est en réalité un membre d'une grande énumération.

Compte tenu du fait que le tag soit aussi une donnée, il peut aussi être converti et reconstitué pour identifier les données qu'il décrit. Afin d'économiser de l'espace mémoire, **un tag** est sur un seul **octet***.

3.2.4 Sérialisation binaire

Possédant un moyen de reconnaître les données avec des **tags**, la sérialisation (conversion) de données KC3 vers un format binaire devient **possible**.

L'implémentation de ce procédé a été pour moi l'occasion d'en apprendre plus sur les procédés d'optimisation mémoire, les formats binaire et hexadécimal* ainsi que l'architecture logicielle/physique des ordinateurs.

3.2.4.1 Format d'encodage

Afin de garantir une conversion de donnée efficace, toute valeur devant être convertie **constitue** un **paquet** (ensemble ordonné) de donnée constituée de son tag et de sa valeur qui se font suite.

La conversion de donnée vers un paquet suit la **structure** ci-dessous :

[Header] 0x0... [Tag]... [Header_binaire] [Taille] [Donnée]

Dans l'exemple ci-dessous nous pouvons observer que les données d'un paquet se suivent. Les valeurs suivent le format hexadécimal à des fins de lisibilité.

Exemple :

	Tag	**Valeur**
Décimal	TAG_INT (= 1)	42
Hexadécimal	0x01	0x2a
Résultat	*...0x01...0x2a...*	

Bien que le format ci-dessus **suffise** pour une **sérialisation** sommaire, la librairie* C qui implémente les fonctionnalités du langage KC3; *libkc3* ajoute une couche de précision supplémentaire en incluant la **taille de la donnée** (*size_descriptor*) dans le paquet.

3.2.4.2 Représentation en langage C

Afin de stocker ces données binaires sérialisées, une zone mémoire à **accès rapide** de taille fixe est réservée et les données sérialisées y sont transférées.

Le langage KC3 supporte entièrement l'encodage **UTF-8***.

Conséquemment, tout **caractère étranger** à l'alphabet latin traditionnel (ex : émoji, kanjis, cyrillique...) peut ainsi être **traité** par le langage et son système de marshalling

En C**,** l'opérateur **sizeof** renvoie la taille d'une valeur (en octets).
Dans le cas général, la taille d'un paquet (sans formattage) se traduit par la formule :

$$\text{total\_size} = \text{sizeof}(\text{TAG\_TYPE}) + \text{sizeof}(\text{size\_descriptor}) + \text{sizeof}(\text{data})$$

Si un des éléments n'est pas présent le paquet est considéré comme invalide et entrainera une perte de donnée durant la désérialisation (voir **section 3.2.5**).

Étant donné que le projet a commencé il y a plusieurs années, cela m'a permis d'apprendre à m'**adapter** à un **environnement** et des **outils déjà existant**, ce qui contraste avec le fait de devoir repartir de zéro à chaque projet de première année a Epitech.

3.2.5 Désérialisation binaire

Maintenant que nous sommes capables de **convertir** des données et de les identifier grâce à un format défini. Nous pouvons procéder à l'étape inverse : les **reconstituer**.

La **désérialisation** binaire consiste en la **reconversion** de données **binaire** en données **interprétable** par une machine dans le contexte d'un langage.

Le développement de la sérialisation et de la désérialisation ont eu lieu **simultanément**. Ce qui a permis une évolution incrémentale des deux systèmes, ces derniers étant complémentaires.

Mon **assignation** portait principalement sur la **désérialisation**, à mesure que le procédé de sérialisation voyait le jour, je devais m'assurer de me synchroniser avec et implémenter sa contre partie. Je rédigeais ensuite des tests pour m'assurer que les données reconstituées soient correctes et modifiait le système d'écriture si besoin.

Ce procédé m'a permis d'**apprendre** des notions de programmation plus poussées et d'approfondir mes connaissances en reconstitution de données.

3.2.5.1 Validation d'un paquet

Lorsqu'un paquet est donné au système de désérialisation, il passe d'abord par une étape de **vérification**.

Le **paquet** est d'abord **scanné** puis est soit accepté soit rejeté.

Les **critères d'invalidation** sont notamment : une corruption du header, un tag inconnu, un header binaire corrompu ou différent de l'original et une taille différente de celle indiquée par le tag.

Les critères étant **nombreux,** l'étape de validation sollicite une **rigueur** d'exécution et d'attention accrue, beaucoup de cas peuvent entrer en conflit entre eux. J'ai néanmoins réussi à créer un outil de validation fiable dans 100% des cas, et ce malgré avoir rencontré beaucoup de difficultés dues au grand nombre cas possibles.

3.2.5.2 Reconstitution d'un paquet

En suivant le format conventionné par **kmx.io**, le système reconstitue les paquets donnée de manière **séquentielle**.

Le header (incluant sa version binaire) et le descriptif de taille sont retirés (sauf exception), ne **reste** plus que le **tag** et la **donnée** elle même. Il suffit alors d'appeler la fonction correspondante au tag obtenu grâce à un **arbre de décision**.

Cette étape fut une des moins contraignantes malgré son chronophagisme.

3.2.5.3 Représentation en langage C

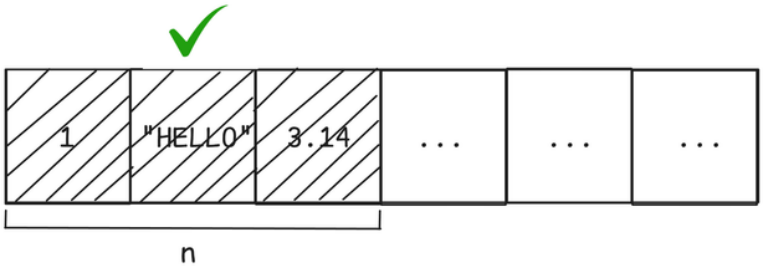
Afin de **convertir** des données dont le type et la taille sont connus,
il suffit de lire *n* octets depuis une région mémoire, *n* étant la taille de la donnée cible.

Dans libkc3, il existe un **ensemble** de fonctions pour lire des données d'un type défini depuis un espace mémoire, ces dernières suivent la convention *"buf_read_[type]".*

```
size_t buf_read(buf *buffer, void *dest, size_t nbytes)
```

Cette fonction lit *n* octets dans un espace mémoire (ici, "*buffer*") et les implante dans dest (destination), c'est cette fonction qui permet de reconstituer une structure de donnée.

Il faut également avancer de *n* octets dans l'espace mémoire temporaire, afin de ne pas lire deux fois la même donnée.



Plus un type est primitif en KC3, plus tôt il a été possible d'opérer dessus. Cette implémentation fut longe, le marshalling de types simples était requis afin de pouvoir créer celui des plus complexes. Suivre un procédé documenté et l'adapter selon les besoins du projet s'est avéré riche en défis techniques.

3.3 Intégration de TLS et SSL

3.3.1 Définition et usage

Le *Transport Layer Security* (TLS) et son prédécesseur, *Secure Sockets Layer* (SSL) sont des protocoles de **sécurisation** des échanges par réseau informatique, notamment par Internet.

Ces protocoles consistent en l'obtention de **certificats** permettant uniquement à deux machines certifiées de communiquer entre elles sans qu'un tiers puisse intercepter leurs échanges, empêchant les attaques de type "*man in the middle**" (homme au milieu).

3.3.2 Fonctionnement

Ces deux protocoles utilisent un **chiffrement** des données avec des algorithmes spécialisés. Chiffrer des donnée revient à les rendre ininterprétables par un tiers n'ayant pas de **clé de décodage**, une des méthodes de chiffrement les plus ancienne est celle de **césar**.

Une machine va alors envoyer des données vers une autre, ces derniers vont se connecter à un port (canal) **sécurisé** par ces protocoles, toute donnée passant par ce canal de communication sera chiffrée par un algorithme avec une **clé** définie puis sera déchiffré par le receveur qui possède un double de la clé.

Ces **paires** de clés sont **uniques**, ce qui assure un hermétisme total.

En combinant cette méthode de chiffrement au marshalling (voir **section 3.2**), nous avons maximisé à la fois la sécurité des données, leur vitesse d'envoi et d'interprétation.

Avant de réaliser ce stage, je ne connaissait pas le domaine de la sécurité Internet, depuis j'ai pu créer la base d'implémentation de ces certificats pour KC3. J'ai également longuement échangé avec mon maître de stage et je comprends maintenant la nécessité de les implémenter dans KC3 et son support web.

3.2.3 Application spécifique à KC3

Non seulement ces protocoles permettent la sécurisation de données, mais ils permettent également le stockage de données marshallisées.

Les interpréteurs **KC3S** et **IKC3** (voir **section 3.5**) tirent profit des certificats **TLS** afin de s'armer de fonctionnalités **RPC** (Remote Procedure Call), qui permettent à deux interpréteurs (client et serveur) de communiquer soit sur la même machine soit sur deux machine différentes.

L'étape la plus **difficile** de cette implémentation fut sans aucun doute la compréhension des **librairies** C spécialisées, notemment **libtls**.

Ma **contribution** à cette fonctionnalité consistait à la rédaction de **tests automatisés**, la recherche et le développement liés à l'ouverture du canal de communication entre interpréteurs, la correction d'erreurs et l'assistance à la conception d'une architecture logicielle robuste.

Les tests rédigés couvrent toutes les commandes développées et comptent combien de tests produisent le résultat attendu, permettant alors de calibrer nos efforts selon les résultats obtenus.

3.4 Écriture de documents au format PDF

3.4.1 Définition

Le *Portable Document Format* (PDF), est un format de document crée par la société Adobe en 1992 permettant d'en conserver la mise en forme.

Ce format s'est rapidement imposé en tant que **format standard** et est devenu une norme internationale pour documents écrits en **2008**. Le présent document suit ce format**.**

Le **PDF** a été conçu pour être supporté par le plus de plateformes possibles, de nos jours, rares sont les machines ne pouvant pas afficher un document au format **PDF**.

3.4.2 Motif d'implémentation

Le langage KC3 a pour dessein et philosophie de s'intégrer dans des plateformes web; la génération de documents est donc une étape incontournable dans son développement.

Il existe de nombreux outils permettant de générer des documents ***.pdf***, cependant aucun ne répondent à un tel besoin d'optimisation appliquée à un langage.

M. Thomas de Grivel désirait dans un premier temps pouvoir créer ses factures sans avoir à faire appel à un logiciel externe, il a donc fait le choix d'implémenter un générateur de documents PDF écrit en KC3 et implémenté grâce au langage C.

3.4.3 Fonctionnement 

Un document **PDF** est un **fichier binaire structuré** composé d'un **en-tête**, d'un **corps**, d'une **table de références** et d'un **pied de page**. Le format est autosuffisant, il peut incorporer lui même des images ou des polices dans des documents en les compressant*.

L'organisation du **PDF** est un graphe, ce qui donne un net avantage à KC3 : la table de références est une base de donnée en KC3 où tous les éléments sont copiés, le header et le pied de pages sont très peu variables.

J'ai eu l'occasion d'**implémenter le support du texte brut** au format PDF pour KC3, j'ai appris a suivre une documentation complexe avec attention afin de respecter une fiche de spécifications techniques et répondre à des standards d'industrie.

3.5 Interpréteurs IKC3 et KC3S

3.5.1 Définition et usage

Un **interpréteur** est un logiciel capable d'extraire et de reconnaître la syntaxe d'un langage dans du texte et d'en exécuter les instructions en temps réel, qu'il soit écrit dans un fichier ou écrit en temps réel dans un terminal.

Donner vie à un langage de programmation revient à **créer un interpréteur** capable de reconnaître un ensemble de mots clés prédéfinis et d'effectuer des opérations en fonction de ces derniers.

L'interpréteur KC3 (**IKC3**) et **KC3 Script** (KC3S) sont les deux interpréteurs capable de donner vie au langage. Tandis que l'un permet d'**exécuter** des instructions en **temps réel** dans un **terminal***, l'autre est utilisé pour **exécuter des programmes complets** écrits dans des fichiers *.kc3*.

De longues discussions sur la théorie des langages de programmation ont eu lieu entre tous les membres du stage, je me suis intéressé aux racines syntaxiques des langages de programmation et je suis maintenant capable d'établir des ensembles de mots clés pertinents pour constituer un langage efficient.

3.5.2 Valeur ajoutée des fonctionnalités

La majorité des fonctionnalités natives aux interpréteurs **IKC3** et **KC3S**, dont certaines susmentionnées sont implémentées dans la libkc3 (à l'exception du **PDF** et de **TLS**), certaines n'ont pas été cités dans le présent rapport car moins pertinentes.

Cette section relate l'impact du stage sur la progression du projet KC3.

3.5.2.1 Marshalling

Comme cité précédemment (voir **section 3.2.2**), le **marshalling** est utilisé pour accélérer le démarrage de **KC3S** et **IKC3** de façon **exponentielle** au travers du chargement de leurs variables d'environnement respectives, ce qui améliore indirectement la **vitesse** de déploiement du site web kmx.io.

Additionnellement, il est maintenant possible d'importer les modules *Marshall* et *Marshall_read* pour faire usage de leurs fonctions associées dans des programmes KC3.

3.5.2.2 Certificats TLS

L'ajout de cette méthode de **sécurisation** permet aux deux interpréteurs de posséder des versions client et serveur se connectant entre eux.

Il est possible de les utiliser soit sur une même machine, soit entre deux machines différentes.

Pour **IKC3**, le client envoie la ligne entrée par l'utilisateur au serveur, qui exécute l'opération et renvoie le résultat de cette dernière au client qui l'affiche à l'écran.

Quant à **KC3S**, le client envoie un fichier au serveur qui va exécuter le programme dans son intégralité et ne rien afficher, sauf si on l'explicite dans le programme.

Cette implémentation permet d'interagir avec des **serveurs web** écrits en KC3 pour leur demander des informations, entre autres.

3.5.2.3 Librairie PDF

Malgré le fait que le **PDF** soit un format extrêmement long à implémenter, particulièrement en langage C, le stage s'est déroulé si bien que le reste des **fonctionnalités** ont été **terminées en avance**, ce qui nous a permis d'implémenter la gestion globale des différents composants du format.

Il est maintenant possible de dessiner et d'écrire dans un document **PDF** afin de rédiger des factures.

La contrainte temporelle a seulement permis d'élaborer une architecture logicielle fiable afin que **M. Thomas de Grivel** puisse reprendre là où nous nous sommes arrêtés.

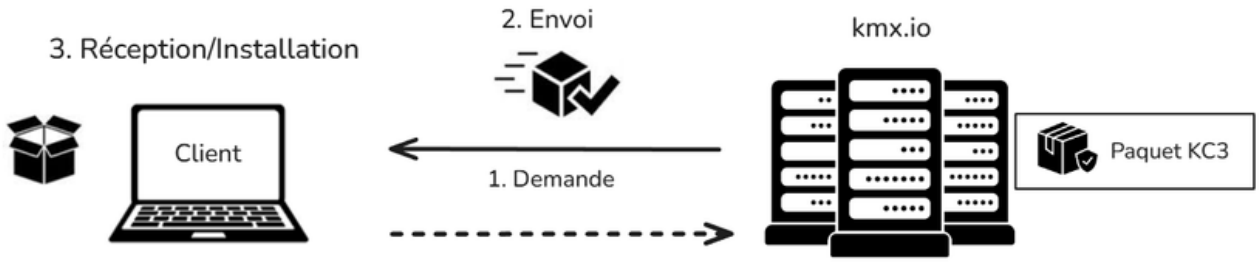
3.6 Package Debian

Un **package** (français : paquet) est un **logiciel installable** par une machine dont le système d'exploitation est issu de la famille ***Linux Debian***.

Un paquet permet aux utilisateurs possédant un système d'exploitation compatible d'**installer KC3** sur leurs machine en une seule commande.

Le paquet est hébergé chez **kmx.io** et est constitué de tous les exécutables déjà compilés* pour cette version de Linux.

L'ajout de cette fonctionnalité est le résultat d'une **suggestion personnelle** formulée à mon maître de stage, j'en suis l'unique créateur.



Ainsi, j'ai pu aborder des thématiques tel que le contrôle qualité, la gestion de version produit et l'amélioration de l'expérience utilisateur.

Cette expérience m'a permis de découvrir comment les logiciels publics se rendent disponible au plus grand nombre.

4. Exigences techniques

4 Exigences techniques

4.1 Prélude

Afin de satisfaire les besoins d'**optimisation**, de **cybersécurité** et de **compatibilité** interplateforme, un certains nombre de principes devaient être respectés.

La norme de formatage du code, l'interdiction/autorisation d'usage de fonctionnalités normalement proscrite ou autorisées à Epitech ont étés les premiers obstacles à franchir afin de s'intégrer pleinement au projet.

Cette section traite des différents **enjeux et contraintes** techniques imposés relatifs à l'accomplissement de la mission.

4.2 Compatibilité interplateforme

4.2.1 Indépendance logicielle

Puisque la **version** de la libraire C standard (libC) **diffère** selon l'architecture processeur et le système d'exploitation, son utilisation est très **restreinte**.

L'intégralité des fonctions usuelles est donc recrée dans la libkc3 afin de fournir les mêmes outils à toutes les machine, peu importe leurs spécificités.

4.2.2 Compilation Conditionnelle

Chaque machine étant différente, la **compilation** des interpréteurs **IKC3** et **KC3S** doit s'y adapter.

La **compilation conditionnelle** consiste à agir ou non sur les instructions destinées à la machine en fonction de **certains critères** définis par le développeur tel que l'**architecture processeur** ou le **système d'exploitation**.

Le **langage C** donne accès à des **informations** sur le système d'une machine via des **constantes** définies ou non.

Ainsi, le produit **exécutable** fini n'est pas exactement le même selon les machines mais ce dernier se **comporte** partout et toujours **de la même manière**.

4.3 Programmation défensive

Dans l’optique de prévenir les **cyberattaques**, le code source du langage KC3 doit assurer une couverture maximale de toutes les erreurs susceptible de se produire.

Ce qui signifie que nous avions pour consigne de causer une **interruption immédiate du programme** si moindre comportement imprévu se manifeste afin de minimiser les risques, ceci étant assuré par une suite de tests automatisés rigoureux.

Selon **kmx.io**, la **fiabilité** d’un logiciel prime par rapport à son efficience mémoire.

5. Conclusion

5 Conclusion

En conclusion, ce stage m'a permis de mettre la pédagogie Epitech en pratique dans des conditions réelles. Grâce aux échanges avec mon maître de stage, ma vision de l'architecture logicielle a mûri, tout comme ma posture de développeur.

Mes objectifs professionnels se sont affinés, me donnant une meilleure vision de mon avenir.

Cette première expérience dans le monde de l'entreprise m'a permis d'améliorer mes compétences existantes mais aussi de déceler mes préférences pour certains domaines informatiques tels que l'analyse syntaxique des langages de programmation et les opérations proches de la machine.

Si les compétences techniques requises pour cette mission portaient sur des domaines que je maîtrisais déjà, ce stage m'a permis de les mobiliser dans un contexte bien plus exigeant et structurant, me préparant aux normes de l'industrie actuelle.

Outre l'aspect technique, j'ai pu saisir les enjeux d'une méthodologie permettant une organisation et une communication efficace au sein de projets techniquement exigeants.

Malgré les difficultés et un manque de séniorité, j'ai su m'affirmer en tant que force de proposition et voir mes compétences être valorisées, ce qui m'a permis d'obtenir la pleine confiance de mon maître de stage pour accomplir ma mission.

Ce stage aura été une occasion de grandir sur bien des aspects, je porte un regard nouveau sur le développement logiciel dans son ensemble. J'ai acquis des compétences qui m'aideront tout au long de ma carrière professionnelle.