HELHa
Haute École Louvain **en Hainaut**

Projet (Q3):

Groupe B

Réalisé par Alexis Zarzycki, Mattéo Castin et Noah Verly en 1BIG.

Pour le cours de, Mr Alary, Mr Steux, Mr Legrand et Mrs Dal

UE IG108 Principes et langages de programmation 2

Année académique : 2021-2022

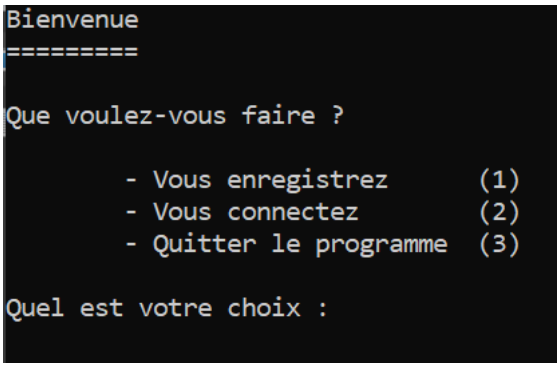
Table des matières :

Présentation du projet et ses objectifs :

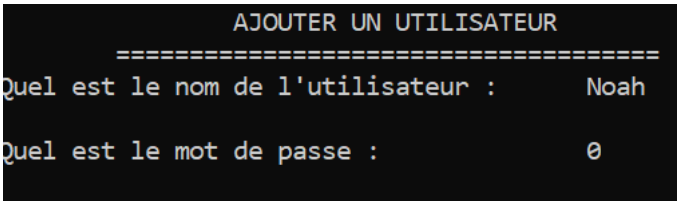
Le projet est comme une sorte de magazine de voitures, que l'on peut consulter soit en affichant toutes les voitures présentes dans la base données, ou bien recherchant un modèle en particulier, mais avant cela, il faudra vous connectez ou vous inscrire avec un nom d'utilisateur unique et un mot de passe qui sera bien sûr chiffré. Il n'y aura qu'un administrateur qui pourra ajouter, supprimer modifier un utilisateur et ajouter de nouvelles voitures. Le projet a pour but, qu'on applique les connaissance apprise en cours, comme savoir crée une base de données correct sur Looping, d'en tirer le script SQL, de savoir gérer des bases de données avec des commandes en c, de pouvoir écrire et lire et naviguer dans un fichier, de savoir utiliser Github et ses commandes, et d'apprendre à travailler en indépendance, tout en restant une équipe et en s'aidant mutuellement.

Mode d'emploi du projet :

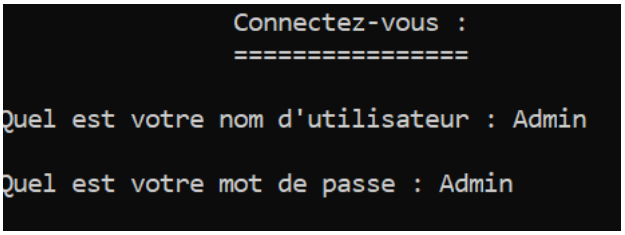
Pour commencer aller dans le dossier source, puis il faut ouvrir le main.exe, vous arriverez sur un petit menu qui vous propose de, soit vous inscrire, soit vous connectez ou bien de quitter le programme.



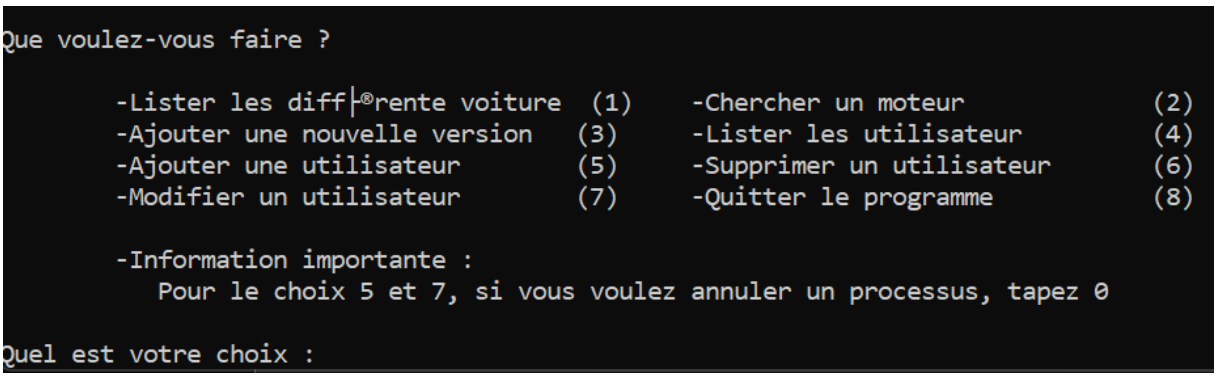
Si vous choisissez de vous inscrire, alors le programme vous demandera de rentrer un nom d'utilisateur UNIQUE, et un mot de passe tout les deux fessant 50 caractères ou moins et si vous voulez annulez rentré(e) : 0 (zéro).



Si vous choisissez de vous connectez, alors le programme vous demandera de rentrer votre nom d'utilisateur et votre mot de passe.

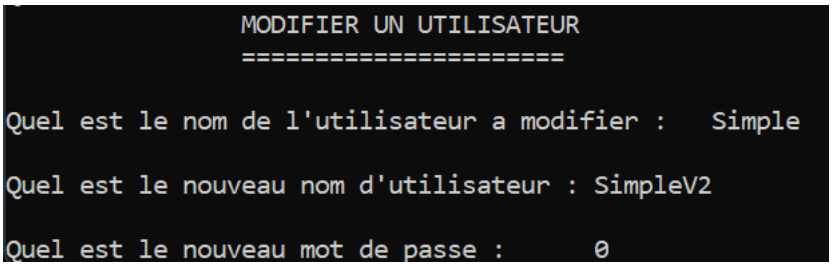


Quand vous serez connecté(e) entant qu'administrateur, vous aurez ce menu, pour utiliser le magazine et gérer les utilisateurs.

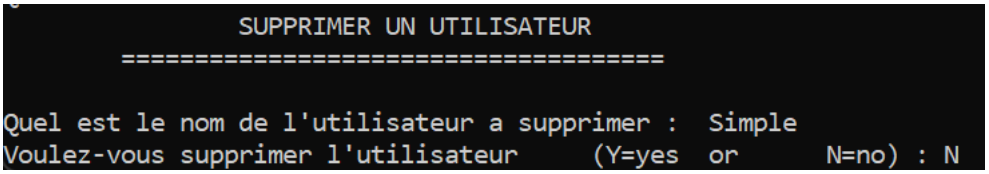


Quand vous choisissez d'ajouter un utilisateur, cela sera le même menu et condition que pour vous inscrire (voir la photo au-dessus).

Quand vous choisissez de modifier un utilisateur, vous arriverez sur un menu, qui vous demandera de choisir l'utilisateur dont vous devez changer les informations si le nom existe, alors le programme vous demandera de choisir un nouveau nom UNIQUE et un nouveau mot de passe qui doivent tout le deux faire 50 caractères ou moins.



Quand vous choisissez de supprimer un utilisateur, vous arriverez sur un menu, qui vous demandera de choisir l'utilisateur que vous voulez supprimer si le nom existe, alors le programme vous demandera de confirmer votre choix en rentrant N (NO) ou Y (YES).



Quand vous choisissez de lister les utilisateurs, le programme vous affichera une liste des utilisateurs avec leur rang.

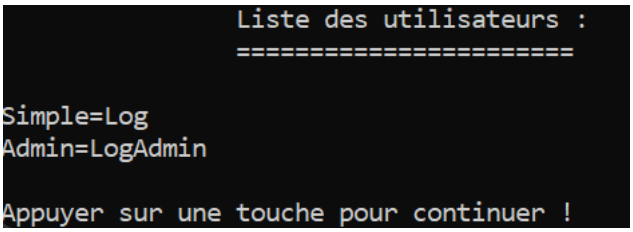


Schéma du projet :

Dans le dossier *.vscode*, il y a les fichiers qui sert à débugger et compiler les fichiers.c pour en faire des .exe qui sont les fichiers qui peuvent être lancés.

Le dossier *database* est le dossier qui contient le fichier *db.loo* qui est le fichier qui contient le modèle de la base de données que nous utilisons.

Le dossier *includes* qui contient les fichiers.c qui sont les fichiers qui contienne le code de toutes les fonctions, les fichiers.h sont ceux qui possède les déclarations de fonctions, la déclarations des #define et #include.

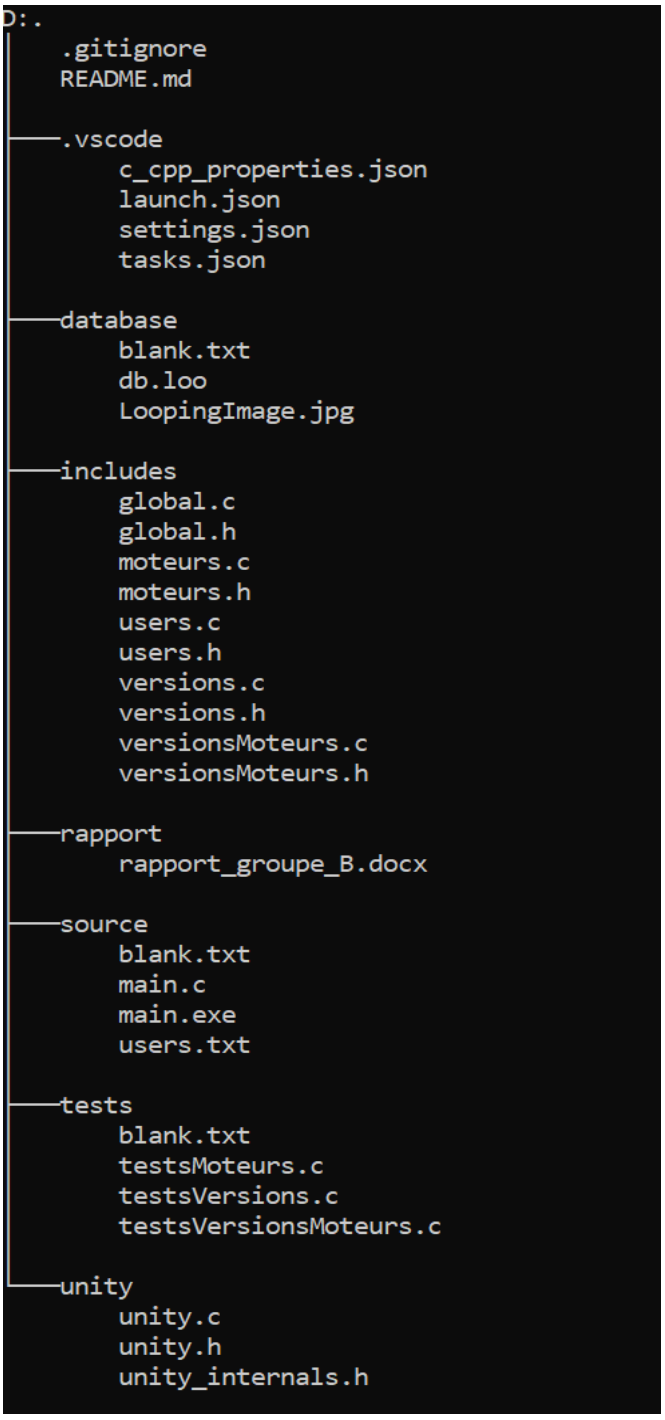
Le dossier *rapport* contient le rapport écrit pour le projet.

Le dossier *source* contient le main.c qui contient le menu principale du programme, son .exe et le fichier avec les informations des utilisateurs.

Le dossier *tests* contient le code des test unitaires demandés. Et pour finir le dossier unity est un le même que le dossier .vscode mais pour les tests unitaires.

Et le *gitignore* est un fichier où l'on indique quel fichier on ne peut pas mettre sur le github.

Capture d’écran de l’arborescence :



Liste des fichiers et leurs buts :

Nom du dossier :	**Nom du fichier :**	**But :**
source	users.txt	Fichier texte contenant les informations des utilisateurs.
database	Db.loo	Fichier qui contient le modèle de la base de données utilisée.
rapport	rapport_groupe_B.docx	Fichier texte qui contient le rapport du projet C
	Rapport_groupe_b.pdf	Fichier pdf qui contient le rapport du projet C

Tableau des fonctions :

Position dans le dossier	**Nom**	**Nom des paramètres et leur type**	**Type de retour**	**Utilité**
include \users.h	lister_uti()	\	Void	Lister tous les utilisateurs présents.
	menu_ajouter(char *rang)	Char *rang	Void	Menu pour rentrer le nom et mot de passe de l'utilisateur.
	ajouter(FILE*fichier,char *log,char *mdp, int dim, char *rang)	FILE *fichier Char *log,*mdp, rang Int dim	Unsigned	Fonction qui vérifie que les paramètres sont bons et les ajoute dans le fichier.
	menu_supprimer()	\	Unsinged	Menu pour choisir qui supprimer.
	Supprimer(FILE*fichier, char *log ,int dim)	FILE *fichier Char *log Int dim	Unsigned	Fonction qui vérifie que le paramètre est bon puis supprime
	menu_modifier()	\	Unsigned	Fonction pour rentrer le nouveau nom et mot de passe d'un utilisateur donné.
	modifier(FILE*fichier, char *log_actu, char * log, char *mdp, int dim)	FILE *fichier Char *log_actu, *log,*mdp Int dim	Unsigned	Fonction qui vérifie que les paramètres sont bons et les modifie s'ils sont bons.
	menu_connexion()	\	Unsigned	Menu pour rentrer votre nom d'utilisateur et votre mot de passe

	Identification(char *log, char *mdp, int dim)	Char *log, *mdp Int dim	Unsigned	Fonction qui vérifie le mdp et le nom d'utilisateur
	check_existes(char *log, int dim, FILE *fichier)	FILE *fichier Char *log Int dim	unsigned	Fonction pour vérifier si un nom existe ou pas et son rang.
	Chiffrement(char *log, char *mdp,int dim)	Char *log, *mdp Int dim	Void	Fonction qui permet de chiffrer un mot de passe.
include \global.h	ErreurFichier(FILE *monFichier, int showText	FILE *monFichier Int showText	Int	Permet d'afficher ou non l'erreur d'un fichier.
	InitConnexion()	/	Void	Permet d'établir la connexion
	closeConnexion()	/	Void	Permet de fermer la connexion
	*TableExist(char *query)	Char *query	Int	Permet de voir si un tableau existe.
	*SqlSelect(char *query)	Char *query	MYSQL_RES	Permet de créer une commande SQL avec le résultat.
	executerCommandeSQL(Char *instructionSql)	Char *instructionSql	void	Permet de créer une commande SQL sans le résultat.
	destroyAllTable()	/	void	Permet de supprimer toutes les tables de données.
	*isDataPresent(char *query)	char *query	void	Permet de voir si une donnée est présente.
include \moteur.h	creationTableMoteurs()	/	int	Permet de créer la table moteur.
	recuperationDesInfosVersions(Char tableauInfosVersions[][51], Int tableauIdVersions[], nbVersions)	Char tableauInfosVersions[][51], Int tableauIdVersions[], nbVersions	void	Permet de récupérer des informations sur une version données.
	creationListePourMoteurs(int idMoteur, int *nbVersions)	int idMoteur, *nbVersions	Void	Permet de créer une liste qui contient les informations sur les moteurs.
	recuperationDuResultat(int tableauIdVersions[], MYSQL_RES *res)	int tableauIdVersions[] MYSQL_RES *res	Void	Permet de récupérer le résultat d'une version données
	*recuperationDesVersions(int idMoteur, int *nbVersions)	int idMoteur, *nbVersions	MYSQL_RES	Permet de récupérer des versions
	recuperationDesInfosMoteurs(char tableauDesMoteurs[][21])	Char tableauDesMoteurs[][21]	void	Permet de récupérer les infos des moteurs dans la DB
	combienDeMoteurs()	/	int	Permet de retourner le nombre de moteur présent dans la DB afin de

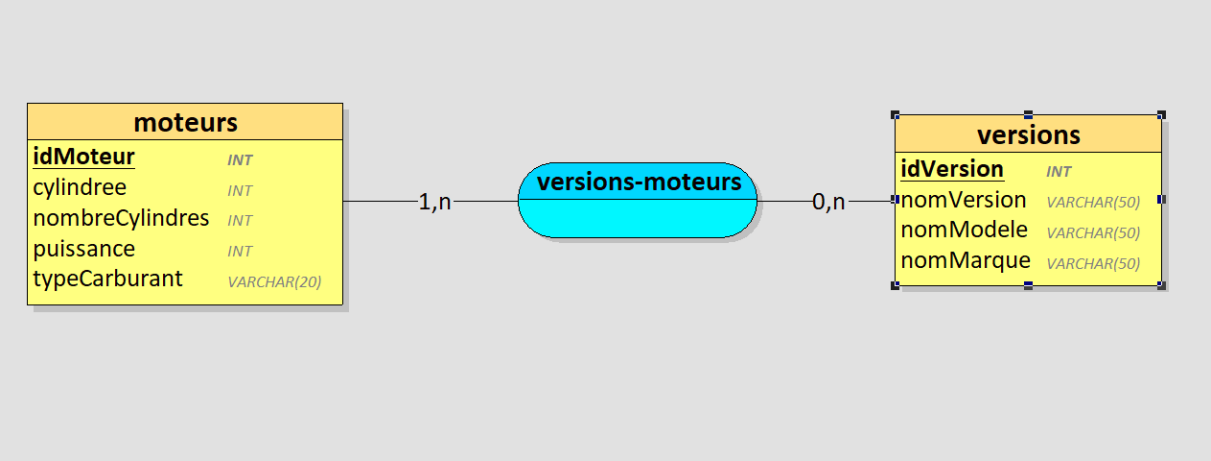
				créer le tableau d'infos moteur à la bonne taille
include \version.h	existTableVersions()	/	Int	Permet de voir si la table version existe
	createTableVersions()	/	Int	Permet de créer la table version
	isTableEmptyVersions()	/	Void	Permet de voir si la table version est vide.
	getVersions(char versions[][100], int *count)	char versions[][100] int *count	Void	Permet d'obtenir un tableau remplis des informations des versions.
	addVersions(char *nameVersion, char *nameModele, char *idMoteur, char *errors)	char *nameVersion, *nameModele, *idMoteur, *errors	Void	Ajouter une nouvelle version
include \version Moteurs.h	creationTableVersionsMoteurs()	/	int	Permet de créer la table versionMoteur

Liste des sources et leur but :

Nom du dossier :	**Nom du fichier :**	**But :**
includes	global.h	Fichier d'en-tête qui contient la déclaration de fonctions assez générales, comme se connecter à la base de données ou la fermer, regarder si une table existe et etc. Il y a aussi la déclaration des include et define.
	global.c	Fichier qui contient le code des fonctions ; connexion, fermeture, table_existe.
	moteur.h	Fichier d'en-tête qui contient la déclaration des fonctions, des includes et les defines par rapport à la gestion de la table moteur du fichier moteur.c.
	moteur.c	Fichier qui contient le code des fonctions ; pour créer la table moteur, récupérer les informations du moteur et la création de la liste des moteurs.
	users.h	Fichier d'en-tête qui contient la déclaration des fonctions, des includes et les defines par rapport à la gestion des utilisateurs du fichier users.c.
	users.c	Fichier qui contient le code des fonctions ; ajout, supprimer, modifier et etc..

	versions.h	Fichier d'en-tête qui contient la déclaration des fonctions, des includes et des defines par rapport à la gestion de la table version.
	version.c	Fiche qui contient le code des fonctions pour vérifier l'existence de la table version, si elle est vide ou pour la créer.
	versionMoteur.h	Fichier d'en-tête qui contient la déclaration de la fonction pour créer la table versionMoteur.
	versionMoteur.c	Fichier qui contient le code pour créer la table versionMoteur dans la base de données.
source	main.c	Fichier principale qui contient le code du menu.
	main.exe	Fichier débugger et compiler du main.c.
tests	testMoteur.c	Test unitaires du fichier moteur.h.
	testUsers.c	Test unitaires du fichier users.h.
	testVersion.c	Test unitaires du fichier version.h.
	testVersionMoteur.c	Test unitaires du fichier versionMoteur.h.

Schéma de la base de données :



Fonction principale :

Nom de la fonction	Fonctionnement
erreurFichier(FILE *fichier, int showText) ;	Cette fonction permet de vérifier qu'il n'y à pas de problème lors de l'utilisation de fichier, nous lui passons le fichier ainsi qu'un argument qui permet de savoir si on doit affiché l'erreur où pas dans la console
initConnexion() ;	Permet de se connecter à la BDD
closeConnexion() ;	Permet de fermer la connexion à la BDD
executerCommandeSql() ;	Permet d'executer une requête sql sans aucune verification
destroyAllTable() ;	Permet de purger la base de données
createAllTable() ;	Permet de créer toutes les tables

Répartition des tâches :

N° de la partie	Alexis Zarzycki	Mattéo Castin	Noah Verly
1er partie	1,2	1,3,4	1
2ème partie	5,7	6	/
3ème partie	10	10	8,9,10

Présentation des test unitaires :

Sources du fichier test	**Sources du fichier testé**	**Nom de la fonction de test**	**Descriptif du test**
Tests\testsMoteurs.c	Includes\moteurs.c	testCreationTableMoteurs	Ce test supprimer l'entièreté de la DB, la récréer en utilisant la fonction creationTableMoteurs et ensuite test si la table moteurs a bien été créer
		testMoteurInexistant	Ce test vérifie que dans le cas où le moteur est inexistant la fonction recuperationDesVersions inscrit bien -1 dans la variable nbVersions
		testMoteurSansVersion	Ce test vérifie que dans le cas où le moteur est ne possède pas de versions la fonction recuperationDesVersions inscrit bien 0 dans la variable nbVersions
		testMoteurExistantEtVersions	Ce test vérifie que dans le cas où le moteur est possède des versions la fonction recuperationDesVersions inscrit bien le nombre de version qui

			est 3 pour ce test dans la variable nbVersions
		testRecupIdVersions	Ce test vérifie que la fonction recuperationDuResultat va bien rechercher tout les IdVersion dans la variable resultat du SELECT
		testRecupInfos	Ce test vérifie que la fonction recuperationDesInfosVersions récupère bien les infos et les stocks à la bonne place dans la matrice
		testCompterLesMoteurs	Ce test vérifie que la fonction combienDeMoteurs retourne correctement le nombre de moteurs attendu
		testRecupInfosMoteurs	Ce test vérifie que la fonction recuperationDesInfosMoteurs récupère bien les infos et les stocks à la bonne place dans la matrice
Tests\testVersionsMoteurs.c	Includes\versionsMoteurs.c	testCreationTableVersionsMoteurs	Ce test supprimer l'entièreté de la DB, la récréer en utilisant la fonction creationTableVersionsMoteurs et ensuite test si la table moteurs a bien été créer
Tests\testsUsers.c	Includes\users.c	Log_pris_admin	Test d'un pseudo d'administrateur qui existe alors la fonction retourne 2
		Log_pris_simple	Test d'un pseudo d'utilisateur qui existe alors la fonction retourne 1
		Log_non_pris	Test d'un pseudo non existant alors la fonction retourne 0
		Ajouter_ok_uti	Test d'ajout d'un utilisateur si il est bien

			ajouter la fonction retourne 1
		Ajouter_ok_admin	Test d'ajout d'un admin si il est bien ajouter la fonction retourne 1
		Ajouter_ko_log	Test d'ajout d'un pseudo déjà existant en admin la fonction retourne 0
		Ajouter_ko_uti_0	Test d'ajout d'un pseudo non valide en admin la fonction retourne 0
		Ajouter_ko_mdp_0a	Test d'ajout d'un mot de passe non valide en admin la fonction retourne 0
		Ajout_ko_uti_long	Test d'ajout d'un pseudo trop long la fonction retourne 0
		Ajouter_ko_mdp_long	Test d'ajout d'un mot de passe trop long fonction retourne 0
		Modifier_ok_admin	Modifier le pseudo d'un admin correctement la fonction retourne 1
		Modifier_ok_uti	Modifier le pseudo d'un utilisateur correctement la fonction retourne 1
		Modifier_ko_log_0	Modifier le pseudo d'un utilisateur avec un pseudo non valide la fonction retourne 0
		Modifier_ko_mdp_0	Modifier le mot de passe d'un utilisateur avec un mot de passe non valide la fonction retourne 0
		Modifier_ko_log_pris	Modifier le pseudo par un pseudo déjà pris la fonction retourne 0
		Modifier_ko_log_long	Modifier le pseudo par un pseudo trop long la fonction retourne 0
		Modifier_ko_mdp_long	Modifier le mot de passe par un mot de passe trop long la fonction retourne 0

		Supprimer_ok	Suppression correct d'un utilisateur la fonction retourne 1
		Supprimer_ko_log	Suppression d'un utilisateur qui n'existe pas la fonction retourne 0
		Supprimer_ko_log_long	Suppression d'un utilisateur trop long la fonction retourne 0
Test\testsVersions.c	Includes\versions.c	testExistTableVersions	Vérification que la fonction retourne bien 1 quand tout est bon
		testNotExistTableVersions	Vérification que la fonction retourn bien 0 quand la table n'existe pas
		testCreateTableVersions	Test que la fonction créer bien la table
		testTableEmpty	Test que la fonction qui vérifie que la table est vide retourne bien 1 quand c'est le cas
		testTableNotEmpty	Test que la fonction qui vérifie que la table est vide retourne bien 0 quand c'est pas le cas
		testTableVersions	Test pour voir si la fonction récupère bien les versions dans la DB
		testAjoutErrorModele	Test d'ajout de version avec un nom de modèle incorrect
		testAjoutIdMotor	Test d'ajout de version avec un idMoteur incorrect
		testAjoutVersionErrorNameVersion	Test d'ajout de version déjà existante

Certaines choses à améliorer :

- Trouver un peu plus d'utilité au système d'utilisateur car en vrai cela est un peu inutile qu'il se fasse un « compte » car tout ce qu'ils font une simple personne aléatoire pourrait le faire. Exemple : Pour effectuer une recherche google il ne faut pas se créer un compte.

- Pourquoi faire en sorte qu'un administrateur puisse rajouter des utilisateurs quand ils peuvent s'inscrire eux même.

Mise en évidence des nouvelles matières :

Pour le cours de MPP, nous avons appris à créer un modèle de base données, de choisir des noms logiques, de faire de bonnes relations. Nous avons aussi appris à écrire du pseudo-code.

Pour le cours de C/LPP, nous avons appris à naviguer, manipuler et écrire dans des fichiers. Nous avons aussi appris à pouvoir gérer une base donnée et la créer avec seulement du code C. A aussi, optimiser notre code, comme faire des choix au niveau de la lecture du code, en mettant par exemple des define, au lieu de mettre 1,0 ou 2.

Et pour le cours de projet, noua avons appris à travailler de façon indépendante, mais en restant une équipe. Nous avons aussi appris à utiliser GitHub et ses commandes. Nous avons aussi dû réaliser des tests unitaires, pour vérifier plus rapidement que nos fonctions font bien ce que nous attendons.

Et en globalité, je dirais que nous avons dû apprendre à être indépendant, ne pas toujours tout demander à mener nos recherches pour trouver une solution, à respecter des deadlines.

Conclusion :

Pour conclure sur ce projet, je dirais que le projet était une bonne épreuve, car nous avons du toucher un peu à tout ce qui touchait au C, que soit des commande SQL pour supprimer, créer, ajouter. Ou bien que se soit dans la manipulation des fichiers, pour gérer nos utilisateurs même si je trouve qu'il y aurait dû en avoir un peu plus.

Ce projet était très bien ce n'était ni trop dur comme celui du Q2 avec le Json, ni trop facile. Pour se travail nous avons d'utilisé toute notre connaissance sur le C, avec bien sur l'aide d'internet pour trouver comment marche certaine fonction ou en trouver de nouvelles qui n'était introduite dans le cours.

Donc pour finir, ceci était une bonne épreuve pour tester nos compétences entant que développeurs C/SQL, gestion du temps, de travail équipe et de travail individuelle.