


Rapport de projet T.E.R

Projet Informatique HLIN405

PunyDuck

Etudiants :

- Valentin FONTAINE
- Paul BUNEL
- Esteban BARON
- Valentin PERON
- Julien LEBARON

Année : 2019-2020

Encadrante : Anne-Elisabeth BAERT

Sommaire

Introduction

Dans le cadre du TER de notre deuxième année à la faculté des sciences de Montpellier nous avons proposé un projet s'intitulant PunyDuck. Le but de ce projet est la réalisation d'une plate-forme de distribution des projets des étudiants du département informatique.

Le groupe de développement est composé de cinq personnes, Valentin FONTAINE, Paul BUNEL, Valentin PERON, Julien LEBARON et Esteban BARON. Nous sommes encadré par Mme Anne-Elisabeth BAERT.

Motivation

Le TER est un module qui apporte beaucoup aux étudiants en gestion de projet ainsi qu'en programmation. Seulement une fois terminés les projets ne sont pas valorisés et tombent dans l'oubli. Notre solution est de proposer une application permettant à chaque étudiants de déposer leurs projets pour les rendre visibles et téléchargeables par tous.

Approches

Notre objectif final étant de produire une application fonctionnelle pouvant être utilisée par tous, nous avions besoin : d'une interface graphique décente, totalement opérationnelle et facilement modulable, ainsi que d'un système de communication efficace entre l'application et une base de donnée. Nous avons donc décidés de diviser le projet en trois partie : tout d'abord la conception d'un framework permettant la réalisation d'interfaces graphiques, puis la réalisation de l'interface graphique elle-même à partir de ce framework, et enfin la partie réseau permettant à l'application de communiquer avec un serveur distant. Ces trois parties communiquent ensembles part le biais d'une architecture logiciel MVC (Modèle-Vue-Contrôleur).
Notre appplication est téléchargeable par tous depuis le site web `https://fvostudio.com/punyduck/`.

Cahier des charges

Le but de Punyduck est de proposer aux étudiants en informatique une plateforme de distribution de leurs différents projets. Cette plateforme devait être une application téléchargeable sur un site internet, et connectée à un serveur pour permettre l'échange de projets entre différents clients.
Pour l'organisation du projet, nous l'avons tout d'abord divisé en plusieurs étapes :

- Mise en place d'un serveur qui servira d'intermédiaire entre les utilisateurs et la base de données.
- Création d'un framework pour faciliter la réalisation de l'application.
- Conception de l'application graphique à l'aide du framework.
- Connexion entre l'application graphique et le serveur.
- Création d'une base de données pour stocker les comptes des utilisateurs et les projets.
- Mise en service d'un site internet permettant le téléchargement de l'application.

Ensuite, une fois les différentes parties du projet dégagées, pour bien définir chacune d'entre elles nous avons décidé du cahier des charges suivant :

Le serveur : Le serveur devra fonctionner de manière asynchrone (nous définiront ce principe dans la prochaine partie). Il pourra héberger de manière sécurisée les données des utilisateurs (nom, mots de passes, ...) et devra être capable de gérer la plupart des erreurs de réseau, comme les coupures de connexion lors d'un téléchargement.

Le framework : Le framework définira un ensemble de classes et fonctions qui serviront de base à la structure d'une nouvelle application. Son utilisation doit être facile avec assez de fonctionnalités déjà disponibles pour pouvoir réduire les appels de fonctions bas niveau. De plus, il devra également permettre la plasticité des interfaces.

L'application : Notre application présentera une interface graphique permettant de naviguer facilement entre différents onglets, de communiquer avec le serveur, et de personnaliser un minimum l'apparence : mode clair / sombre, position de certains éléments de la fenêtre, couleur de l'arrière plan.

La base de données : Elle stockera les données relatives aux comptes des utilisateurs ainsi que les projets. Les données confidentielles (mots de passe) seront cryptées.

Le site : Il sera constitué d'une unique page permettant de télécharger l'application. Il possèdera un design « responsive » et dynamique.

Partie 1

Technologies utilisées

1.1 Langages

Pour la programmation de notre application, nous avons choisis assez naturellement d'utiliser le langage C/C++. En effet, c'est le langage que nous avons le plus étudié à l'université, avec lequel 3 des 5 membres du groupe avaient codé leur projet de Licence 1 CMI, et qui nous offrait un support puissant et efficace pour notre interface graphique.
De plus le C++ nous a permis de coder en orienté objet, ce qui était une nécessitée pour la création de notre application. En effet, notre programme utilise énormément ce type de programmation : nous développeront ce point plus tard dans la partie 3 sur le développement logiciel.

Pour l'interface graphique, nous avons utilisé la bibliothèque OpenGL : une interface regroupant environ 250 fonctions différentes qui peuvent être utilisées pour afficher des scènes tridimensionnelles complexes à partir de simples primitives géométriques. Du fait de son ouverture, de sa souplesse d'utilisation et de sa disponibilité sur toutes les plates-formes, OpenGL est une des bibliothèque graphiques les plus utilisée par la majorité des applications scientifiques, industrielles ou artistiques 3D.
L'utilisation de cette bibliothèque (associée à GLFW pour la gestion des fenêtres) était donc un choix assez basique puisqu'elle est très répandue, mais nous avons dû apprendre à l'utiliser : celle-ci est beaucoup plus complexe que les librairies que nous avions utilisées auparavant, comme SFML ou PyGame, de par son bas niveau.

Pour gérer la partie réseau de notre application, nous n'avions pas besoin d'utiliser un langage puissant comme le C/C++, nous pouvions donc nous rabattre sur un langage moins performant mais plus facile d'accès. De ce fait, nous nous sommes dirigés vers le langage python, qui est un langage très simple d'utilisation, pratique pour débuter dans le domaine complexe de la programmation que représente la programmation en réseau. De surcroît il est plutôt efficace pour gérer les entrées sorties, notamment pour la lecture et l'écriture dans des fichiers, qui est une notion centrale dans notre projet.

Cependant, le langage Python de base ne permet pas de faire de la programmation en réseau, nous avons donc du choisir une des nombreuses bibliothèques disponibles permettant de faire de la programmation réseau en Python. Après réflexion, nous avons opté pour le module « asyncio » pour deux raisons : premièrement, ce module offrait une interface de programmation en réseau de haut niveau donc plus simple d'utilisation, ce qui nous arrangeait particulièrement étant donné que nous sommes parfaitement novices dans ce domaine ; Deuxièmement, le gros point fort de ce module est le fait qu'il implémente une nouvelle manière de programmer : la programmation asynchrone. La programmation asynchrone pour les entrées/sorties est une forme de programmation parallèle permettant d'exécuter d'autres parties d'un programme lorsque celui-ci est en attente d'une transmission de donnée, afin de grandement diminuer le temps d'exécution du programme.

1.2 Outils

Pour réaliser notre projet nous avons utilisé des outils différents et spécifiques à chaque tache.

Nous avons utilisé le système de gestion PostgreSQL [1] pour gérer la base de donnée de notre serveur, ce système nous permettant d'utiliser des requête SQL en Python et ainsi de mettre en application les connaissances acquises cette année avec le module HLIN304.
Pour pouvoir exécuter des requêtes SQL depuis le Python, nous avons employé le module psycopg2 [2]

Pour la partie modélisation de l'application nous avons opté pour le Langage de Modélisation Unifié (UML) vu en cours, dans le module HLIN406.

Toutes les communications du groupe ce sont faites sur le logiciel Discord [3], un logiciel facilitant grandement les communications en groupes avec par exemple le partage d'écran, les groupes vocaux et textuels, le fait de pouvoir partager des morceaux de code directement dans le canal de discussion textuel etc..

Pour ce qui est du partage et des sauvegardes du code, nous avons utilisé Git (un logiciel de gestion de versions décentralisé) via un serveur GitHub [4] qui nous a permis de garder nos anciennes versions, de ne rien perdre en cours de route, et de pouvoir partager l'avancée du projet avec les autres étudiants du groupes et notre encadrante.

Pour ce qui est de l'éditeur de code utilisé, nous nous sommes tous penché sur Visual Studio Code [5] pour sa fiabilité et sa mise en page agréable. De plus, nous avons pu grâce à ce logiciel coder à plusieurs en même temps avec sa fonctionnalité de partage en temps réel.

Enfin, nous avons utilisé le langage LaTeX via la plateforme Overleaf [6] pour rédiger ce rapport.

1. PSQL : `https://www.postgresql.org/`
2. psycopg2 : `https://www.psycopg.org/`
3. Discord : `https://discordapp.com/`
4. GitHub : `https://github.com/`, notre projet : `https://github.com/valfvo/punyduck`
5. VSCode : `https://code.visualstudio.com/`
6. Overleaf : `https://www.overleaf.com/`

Partie 2

Développement logiciel, Conception, Modélisation, Implémentation

2.1 Présentation du développement

Le développement du projet est séparé en trois parties :

- le framework de l'interface graphique ;
- l'interface graphique utilisant le framework ;
- la communication réseau entre le client et le serveur.

Le développement de l'application s'est déroulé en plusieurs étapes qui vont être développées points par points.

2.1.1 Apprentissage de l'utilisation d'OpenGL

La première étape fut d'apprendre l'OpenGL, qui fut difficile à comprendre et à utiliser de par son bas niveau comparé aux librairies qu'on a l'habitude de voir en L1 et L2.
La première partie de l'apprentissage d'OpenGL a été de pouvoir dessiner un triangle d'une certaine couleur. Puis grâce à ce triangle de pouvoir dessiner un rectangle. Ainsi de suite nous avons pu dessiner toutes sortes de formes, tel que les cercles par exemple. La deuxième partie a été de pouvoir coller des textures sur ces surfaces précédemment créées. Cette partie n'a pas posé réellement de problème. Pour la troisième partie, nous devions pouvoir mettre une surface dans une autre surface, comme dans la figure 2.1. Ceci était primordial pour la réalisation de l'application graphique car cette simple tache à permis la superposition des éléments graphiques de l'application.

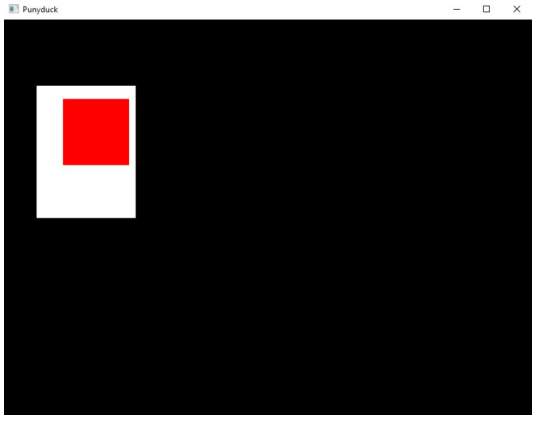


FIGURE 2.1 – Capture d'écran de notre premier essai d'une superposition de surfaces

La dernière partie était de créer le texte, c'est-à-dire de pouvoir afficher du texte à l'écran.

La seconde étape du développement de l'application a été de modéliser en UML les différentes parties de l'application et ses mécanismes.
Pour commencer, il fallait trouver un système permettant de lier des éléments entre eux afin de pouvoir les superposer, en créer des nouveaux à partir d'anciens, de supprimer un élément et ses sous éléments qui le compose etc.. C'est pour cela que nous avons créé les "Quarks", un système où un élément a un père et n fils (où n appartient à l'intervalle $]0, +\infty[$) et a accès à ses frères. Grâce à ce mécanisme, tous les problèmes cité précédemment ont été résolus. Ce système a donc été la base de toute l'application.
Ensuite nous avons créé les différentes classes qui allaient nous servir pour l'implémentation. Comme par exemple :

- "LQTexture" qui génère les textures
- "LQSurface" pour les surfaces
- "LQViewable"
- "LQColor" pour la couleur
- "LQNumber"
- "LQText" pour le texte
- "LQButton" pour les boutons

La troisième étape fut la phase d'implémentation. D'une part nous nous sommes occupés des classes en rapport avec d'OpenGL tels que "LQShader", "LQTexture" et "LQSurface". Par la suite nous avons coder les différentes classes en suivant l'UML précédemment réalisé.

2.1.2 Création de l'interface graphique

L'interface graphique a été créée en plusieurs temps. Dans un premier temps il a fallu trouver une représentation graphique validée par tous les membres du groupe. Pour cela, nous avons fait la liste des différentes pages qui seraient dans l'application :

Connexion : une page permettant de s'inscrire et de se connecter à son compte ;

Projet : une page affichant tous les projets stockés sur le serveur, où on peut également les télécharger ;

Dépôt : une page pour pouvoir déposer ses projets sur l'application.

Puis on les a dessinés sur papier une par une pour que chaque membre du groupe ait une vision très précise de l'aspect visuel de l'application. Enfin, pour l'implémentation des pages nous avons utilisé le framework créé dans ce but. Il a fallu créer les différents composants de la page pour au final les assembler et former la page souhaitée (une page correspondant à une vue dans notre architecture MVC).
Grâce au framework, on a pu lier toutes les pages (vues) entres elles et permettre ainsi la navigation entre les différentes pages de l'application. Par exemple pour la page "Projet", il a fallut créer cette suite d'éléments qui la composent :

- Une liste de bouton permettant de choisir comment trier les projets ("mieux notés", "plus récents"...) [1]
- Une barre de recherche pour accéder à un projet en écrivant son nom
- Des boutons qui permettent de trier les projets sous forme de liste ou sous forme de mozaique avec les images des projets[1]
- Les projets composés d'un nom, d'un "tag" (jeu, algorithme, application...) et d'une image

1. Cette fonctionnalité n'est pas encore implémentée, c'est donc pour l'instant une simple image

Une fois créés, nous avons du les assembler pour former la page. Pour cela, la classe "LQTreeCreator" du framework nous a grandement facilité la déclaration des composant et permis une meilleure structure pour la création des pages. C'est une structure d'arbre, chaque composant est un "Quark" qui a une référence sur son père, son précédent frère (prevSibling) et sur son prochain (nextSibling). De plus, on peut accéder à la position des composants avec des méthodes spécifiquement créées telles que top(), right(), bottom(), left(). Ainsi nous avons pu placer et créer des composants en fonction de leur père ou des composants précédent ou suivant.
Lors de la création des pages avec les composants, nous devions avoir accès au père et aux frères et aux fils. Pour cela les méthodes sub() et super() ont été créés. La méthode sub() permet d'accéder au fils de ce composant, amenant donc la possibilité de créer de nouveaux éléments à partir d'ancien déjà créés. Une fois le composant souhaité créé, nous pouvons vouloir revenir sur le composant père qui se trouve le plus souvent être la vue en elle-même. Pour cela, on fait appel à la méthode super().
Cette structure doit nous faire penser à une structure d'arbre où l'on peut fixer des branches (composant) sur la racine ou sur la branche où l'on se situe.c'est-à-dire créer des branches à partir d'autres branches existantes et fixer cette nouvelle branche à un objet (soit la racine, soit la branche sur laquelle on se situe).

2.1.3 Communication réseau et base de donnée

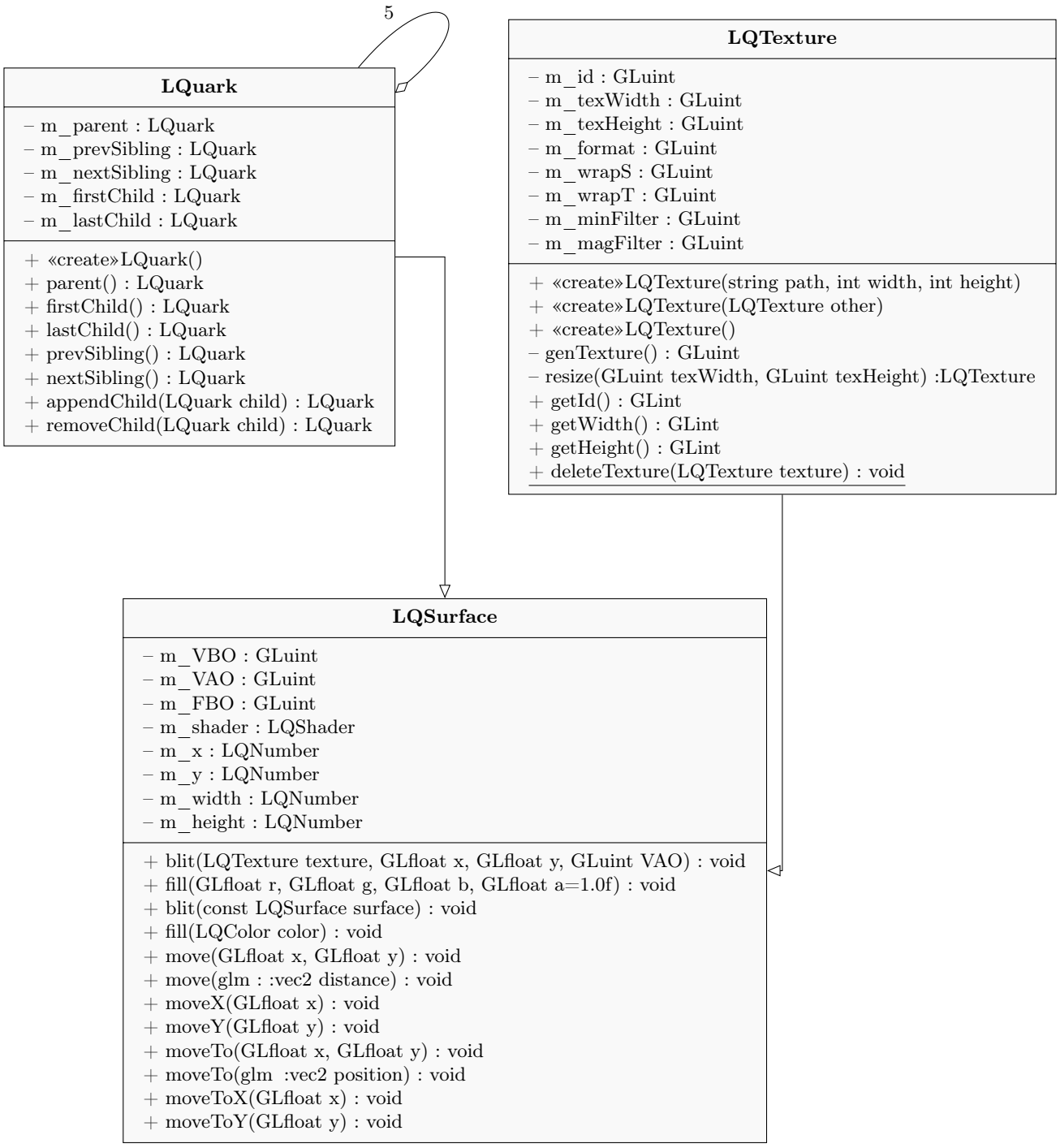
Réalisation d'un client et d'un serveur

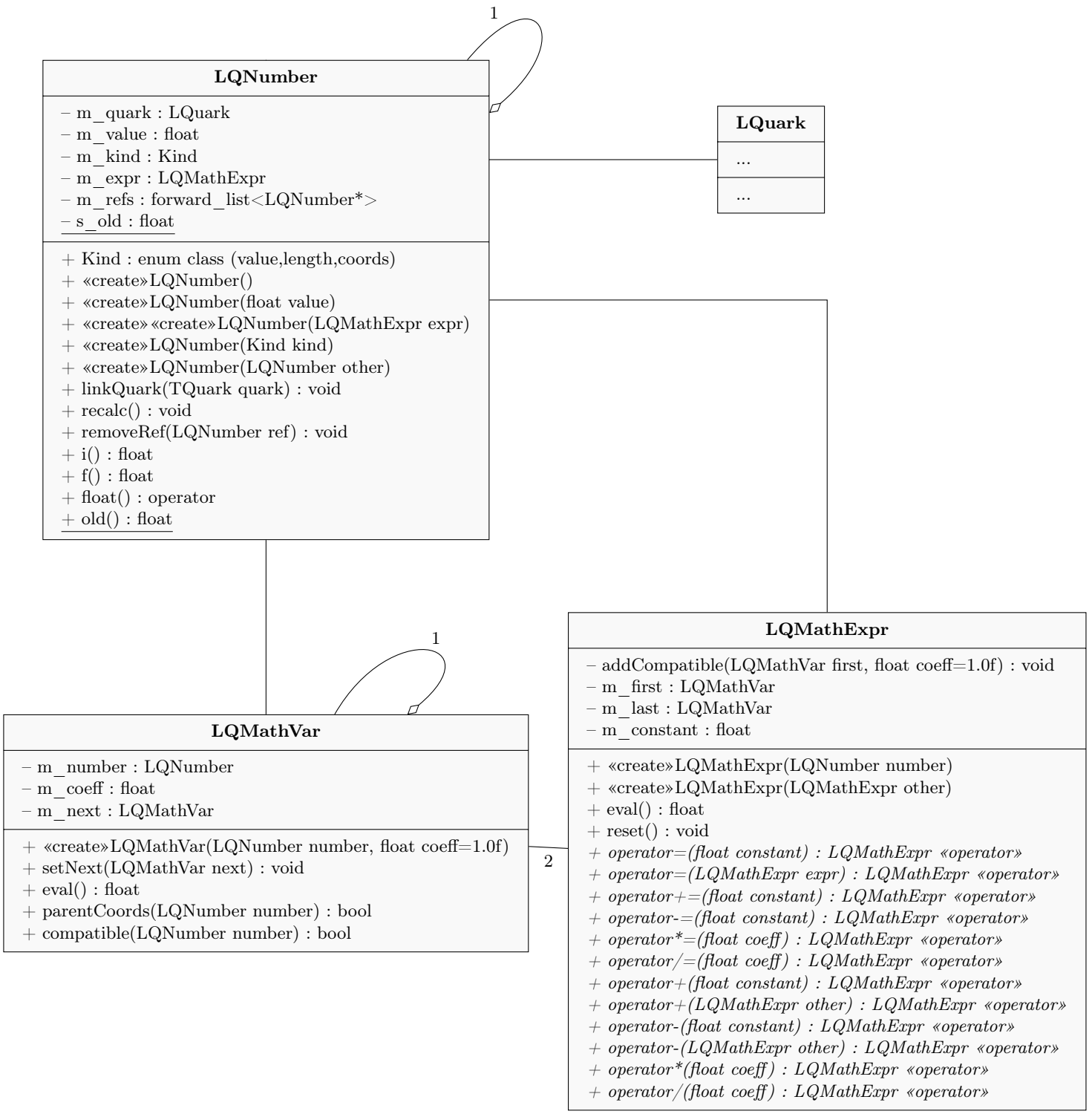
La partie réseau de notre application s'est faite à part : en effet, elle utilisait un langage différent du reste (le Python) et devait s'exécuter dans un thread secondaire. Cette partie a donc commencé par l'apprentissage de la programmation en réseau via le module asyncio de Python. Ce module étant relativement simple d'utilisation, cette partie n'a pas posé beaucoup de problèmes.
Une fois que les bases étaient maîtrisées, nous avons dû créer deux programmes communiquants : un client et un serveur. Le rôle du client était de transmettre les requêtes de l'application au serveur, qui lui devait les analyser, et envoyer une réponse adaptée au client, qui la transmettait à nouveau à l'application. Les requêtes que le serveur devait être capable de gérer étaient peu nombreuses : transfert de projet dans un sens ou dans l'autre (client vers serveur ou serveur vers client) avec ajout dans la base de donnée, inscription, connexion, et demande de données à la base de données. Ainsi, nous avons opté pour une solution extrêmement simple : pour chacune de ces cinq requêtes, nous associons un numéro (de 1 à 5), que nous insérons au début de la chaîne-requête au moment de sa création dans le C++. Une fois la chaîne envoyée au client python, celui-ci lit le premier octet (donc le numéro de la requête) et exécute la fonction associée. Dans le même temps, il envoie la chaîne au serveur qui fait exactement la même chose. Ainsi, chaque partie de notre système client-serveur sait ce qu'elle doit faire en fonction du besoin de l'application.

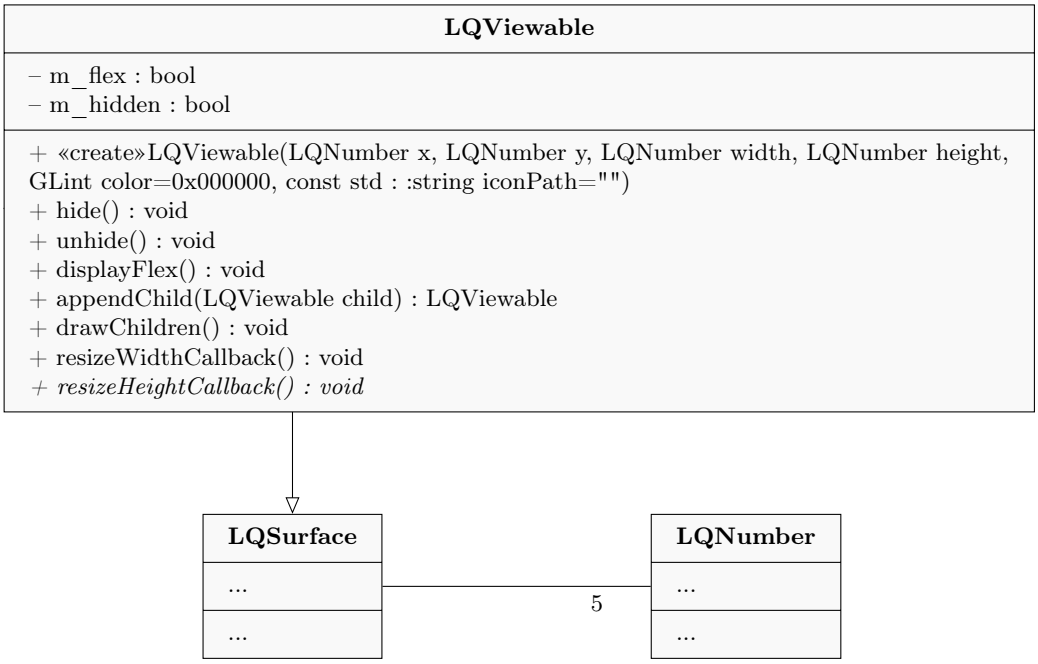
Interface Python/C++

Une fois le client et le serveur finis et leur fonctionnalités implémentées, nous devions faire communiquer le client avec l'application C++ afin qu'ils puissent se transmettre des informations. Pour cela, nous avons créé en C++ un module Python basique nommé « gateway » utilisant une classe ClientGateway. Cette classe contient deux attributs : deux std : :queue de tableau de charactères, l'une stockant les requêtes à envoyer au serveur, l'autre stockant les réponses (m_requests et m_responses).
Ainsi le module possède deux fonctions : « poll_request » qui retourne le premier élément de m_requests, et « transmit_response » qui remplit m_responses avec la chaîne passée en paramètre. Le client peut donc communiquer avec l'application de la manière suivante : à chaque tour de boucle, il appelle poll_request pour voir s'il a reçu une action à exécuter, puis en cas de réponse du serveur, il transmet les informations nécessaire grâce à transmit_response. De la même manière, lorsque l'application veut communiquer avec le client, elle remplit m_requests ou regarde le contenu de m_responses.
Enfin, étant donné que l'application n'a pas accès directement à m_requests et m_responses puisque que le module s'exécute dans un second thread, nous déclarons une instance de ClientGateway dans le contrôleur, puis nous exécutons dans le second thread la fonction de la classe qui lance le programme Python du client. Cette opération nous donne donc accès aux attributs de la ClientGateway du second thread directement depuis le contrôleur, dans le premier thread.

2.2 Modélisa tion




1
LQNumber
– m_quark : LQuark
– m_value : float
– m_kind : Kind
– m_expr : LQMathExpr
– m_refs : forward_list<LQNumber*>
– s_old : float
+ Kind : enum class (value,length,coords)
+ «create»LQNumber()
+ «create»LQNumber(float value)
+ «create»«create»LQNumber(LQMathExpr expr)
+ «create»LQNumber(Kind kind)
+ «create»LQNumber(LQNumber other)
+ linkQuark(TQuark quark) : void
+ recalc() : void
+ removeRef(LQNumber ref) : void
+ i() : float
+ f() : float
+ float() : operator
+ old() : float
LQuark
...
...
1
LQMathVar
– m_number : LQNumber
– m_coeff : float
– m_next : LQMathVar
+ «create»LQMathVar(LQNumber number, float coeff=1.0f)
+ setNext(LQMathVar next) : void
+ eval() : float
+ parentCoords(LQNumber number) : bool
+ compatible(LQNumber number) : bool
2
LQMathExpr
– addCompatible(LQMathVar first, float coeff=1.0f) : void
– m_first : LQMathVar
– m_last : LQMathVar
– m_constant : float
+ «create»LQMathExpr(LQNumber number)
+ «create»LQMathExpr(LQMathExpr other)
+ eval() : float
+ reset() : void
+ operator=(float constant) : LQMathExpr «operator»
+ operator=(LQMathExpr expr) : LQMathExpr «operator»
+ operator+=(float constant) : LQMathExpr «operator»
+ operator-=(float constant) : LQMathExpr «operator»
+ operator*=(float coeff) : LQMathExpr «operator»
+ operator/=(float coeff) : LQMathExpr «operator»
+ operator+(float constant) : LQMathExpr «operator»
+ operator+(LQMathExpr other) : LQMathExpr «operator»
+ operator-(float constant) : LQMathExpr «operator»
+ operator-(LQMathExpr other) : LQMathExpr «operator»
+ operator*(float coeff) : LQMathExpr «operator»
+ operator/(float coeff) : LQMathExpr «operator»



2.3 Fonctionnalités de l'interface

L'interface graphique est composé de plusieurs fenêtres, chacune spécifique. D'autant plus que l'ensemble peux être personnalisable (placement de certains bloc présent sur l'interface). Pour la partie commune à chaque fenêtre de l'interface on a la barre des onglets contenant, dans cet ordre :

- "L'accueil", comprenant les lancements rapide d'application dernièrement utilisées ainsi que les nouveautés lié à l'application ou de certains projets.
- Les "Projets", où l'ensemble des projets mis en ligne seront affichés.
- Le "Profil" est l'endroit où l'on peut modifier ses données personnelles, sa page de profil et les paramètres de l'application.
- Le "Dépôt" vas servir à faire une demande de dépôt de projet, afin qu'il soit vérifier avant mise en ligne (pour éviter les dépôt de virus ou autre).
- Icônes basique tel que épinglé, réduire et fermer. Mais contient aussi le mode nuit (fond clair qui devient foncé).

La fonctionnalité principale étant d'aller sur la page projet, voir un qui nous intéresse ou faire une demande de dépôt pour notre projet. Maintenant cas par cas voyons les différentes interactions.
Pour l'accueil il est possible de cliquer sur les articles mis en avant enfin de pouvoir les lires en détails, mais aussi les lancements rapides des derniers projet lancer en cliquant sur l'icône en question. Pour les projets, les fonctionnalitées qui changes sont les touches de tri et d'affichage ainsi que la barre rechercher. Pour la collection, c'est pareil sauf que l'on peux acceder à un menu d'action a coté de chaque projet (ressemblant à 3 petit points) :

- Désinstaller
- Emplacement
- Page projet
- Désabonner / Ne plus suivre

Ainsi que les boutons d'action, lancer qui vas démarrer l'application, et installer qui vas télécharger et installer l'application. En ce qui concerne la page de présentation du projet, on peux interagir avec la notations (données une notes ou envoyé un commentaire) mais également un lien qui va envoyé vers un forum existant en lien avec le projet. On a notamment accès au bouton suivre pour l'avoir dans notre collection et le bouton j'aime pour aimé un projet.
Les interactions avec l'interface concernant le profil sont, la modification de fond et de l'image de profil ainsi que des autres paramètres (nom, prénom, mot de passe,...), l'accès au paramètre du client (langue,emplacement,...) et le bouton de déconnexion.

2.4 Statistique

Partie 3

Algorithmes et Structures de Données

3.1 Présentation des principales structures de données

3.2 Présentation des principaux algorithmes

3.3 Complexité théorique

Partie 4

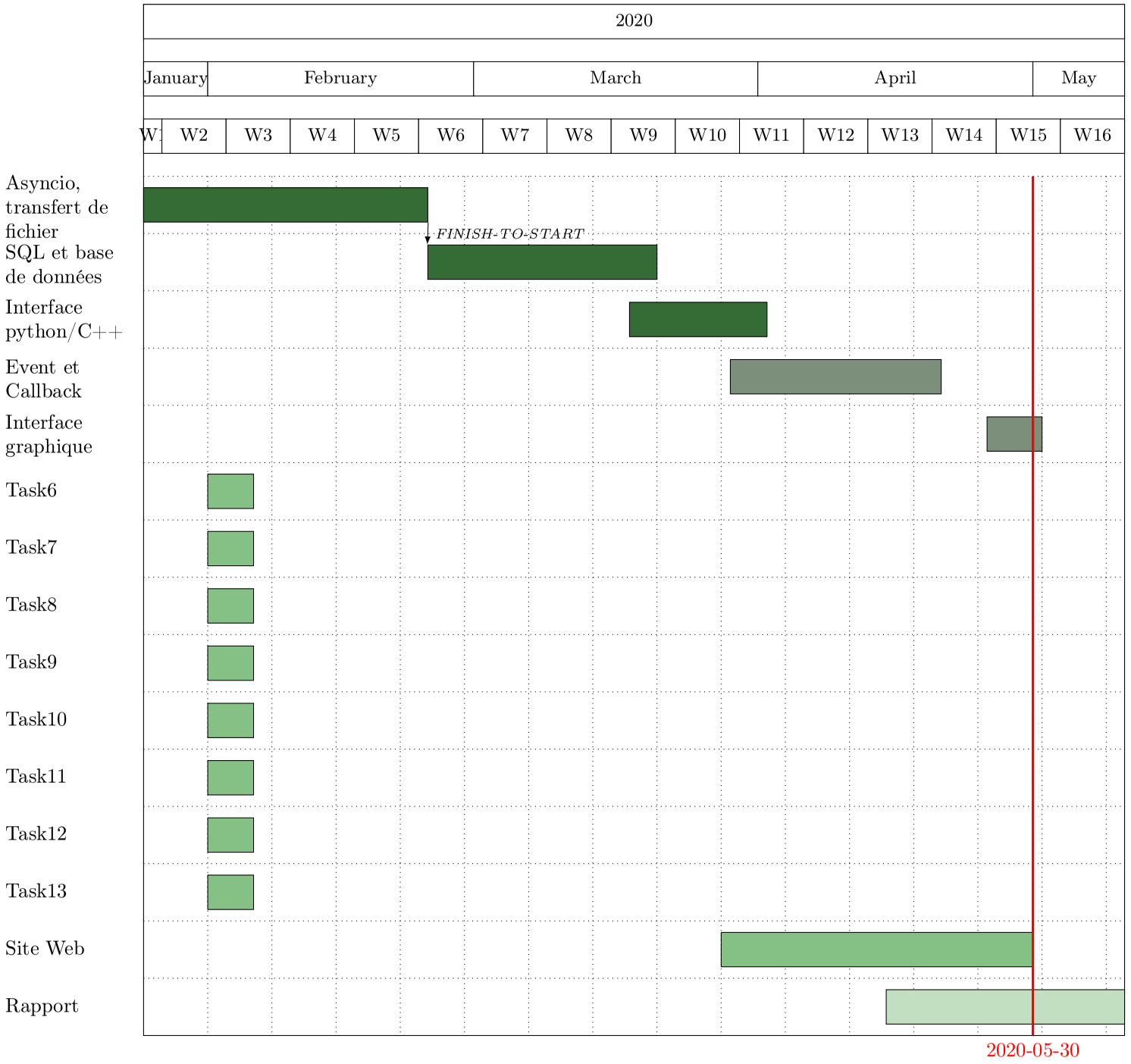
Gestion du Projet

4.1 Organisation et planification

Durant le développement de Punyduck, dès son début, on a décidé de se mettre dans le projet après le repas du midi, ou dès que l'occasion ce présentait. Étant toujours ensemble la plus part du temps, c'était donc de manière quotidienne que l'avancer du projet ce faisait. Quand le travaille ce faisait à distance via la plate-forme "discord", on faisait en sorte de garder toutes traces de ce qui avait été fait, et de ce qui restait à faire, ainsi que bien penser à mettre le git à jour pour chaque membres du projet.

À chaque fin de mois du projet, une entrevu avec notre encadrante Mme Anne-Elisabeth BAERT afin de faire le point sur l'avancer du projet et de connaître ses priorités. Cela nous à permis de ne pas dériver et d'arriver jusqu'à l'aboutissement du projet.

Le développement du projet à été découpé en trois partie (hors site web et rapport) qui sont, la partie réseau en python (le serveur du client, la base de données SQL et ses requêtes), la partie framework et frontwork. Le groupe à été séparer en deux groupe soit un pour la partie réseau et l'autre pour la partie framework. La dernière partie pour le frontwork a réuni les deux groupes afin de produire la liaison Client / Serveur et la mise en place de l'interface graphique grâce au framework. Également, le site web se faisant en parallèle du projet par un des membres du projet durant cette phase.


2020
January
February
March
April
May
W1
W2
W3
W4
W5
W6
W7
W8
W9
W10
W11
W12
W13
W14
W15
W16
Asyncio, transfert de fichier
SQL et base de données
Interface python/C++
Event et Callback
Interface graphique
Task6
Task7
Task8
Task9
Task10
Task11
Task12
Task13
Site Web
Rapport
FINISH-TO-START
2020-05-30

Partie 5

Bilan et Perspectives

Le bilan de cette fin de projet est très positive, en comparant à notre cahier des charges, la quasi totalité des demandes et des objectif a été atteints. Malgré le manque de mains d'oeuvres pour la partie informatique, 3 au lieu de 5, le projet à pu voir le jour avec succès. Bilan partie réseau Bilan partie front et framework Bilan des aides (cours, site, grâce à quoi, etc...)

Conclusion

Conclusion globale, point négatif et positif + terminer par les perspectives du client.

Partie 6

Bibliographie et Annexes