Équipe 204

Projet d’évolution d’un logiciel
Plan de projet

Version 1.2

Historique des révisions

Date	Version	Description	Auteur
2021-02-10	1.0	Début du document	Antoine Morcel Étienne Plante Simon Malouin Samuel Poulin Maxime Fecteau Guillaume Beausoleil
2021-02-14	1.1	Continuation du premier jet	Guillaume Beausoleil Samuel Poulin
2021-04-15	1.2	Révisions du document	Antoine Morcel Maxime Fecteau

Table des matières

Plan de projet

1. Introduction

Le plan de projet décrit sommairement la solution proposée pour la transformation de PolyDessin. De plus, il contient les mécanismes de gestion et de suivi utilisés à travers le projet, un échéancier, une description de l'équipe de développement et l'entente contractuelle.

2. Énoncé des travaux

2.1. Solution proposée

Nous allons développer une application qui est un jeu multijoueur de dessin. L'application s'appellera PolyDessin. L'application permettra de jouer seul ou avec des amis à un jeu de dessin où il faut deviner ce que l'autre dessine. Afin de rendre notre jeu accessible à travers une application desktop ou mobile, nous avons choisi d'utiliser le cadriciel Electron ainsi que le langage de programmation Kotlin. En ce qui concerne l'application desktop, Electron nous permettra d'avoir comme point de départ notre projet de deuxième année (PolyDessin) puisque ce cadriciel rend possible l'exécution d'applications web dans un environnement desktop. Quant à l'application mobile, le langage de programmation Kotlin nous permettra de créer une application Android native ayant toutes les fonctionnalités requises.

Afin de permettre le jeu en mode multijoueur, nous avons choisi d'utiliser un serveur express propulsé par Node.js. Cette implémentation est bien documentée et répond à la majorité des besoins du projet. Afin d'offrir des fonctionnalités en temps réel, la librairie Socket.IO permettra une communication par websockets entre les différents clients. Finalement, nous utiliserons une base de données MongoDB pour le stockage des informations des utilisateurs du jeu.

2.2. Hypothèses et contraintes

Pour la réalisation de l'application, on pose l'hypothèse que le matériel informatique est suffisamment performant pour pouvoir utiliser les logiciels de développement tels Electron, Kotlin, les librairies du projets ainsi qu'effectuer de la communication avec le serveur. On prend aussi en compte que les différentes librairies externes utilisées dans le projet tel Socket.IO sont maintenues et fonctionnelles.

On pose comme hypothèse, que pour la durée totale du projet, l'équipe de 6 développeurs est disponible et n'a pas d'empêchement puisque la charge de travail est répartie à travers les sprints pour une équipe de 6.

Au niveau des contraintes, nous avons un échéancier, présenté plus loin dans le plan de projet, sur lequel nous nous basons pour les dates limites d'implémentations des fonctionnalités. Cet échéancier suit le processus de développement de l'application pour permettre la création d'un prototype

Comme contrainte matérielle, on prend en compte que le serveur de l'application est disponible et pris en charge de manière permanente sur un serveur personnel ou un service en ligne tel AWS ou Azure pour permettre l'utilisation de l'application.

2.3. Biens livrables du projet

Une première livraison des artefacts a lieu le 19 février incluant le plan de projet, le SRS, la liste des exigences, le document d'architecture logicielle et le protocole de communication. De plus, à la même date, nous devrons livrer un prototype qui implémente un système de messagerie. La deuxième et dernière livraison d'artefact aura lieu le 19 avril incluant une mise à jour des artefacts remis à la première date et en plus le plan des tests et les résultats des tests, ainsi que la livraison finale de l'application.

3. Gestion et suivi de l'avancement

3.1. Gestion des exigences

Pour ce qui est de l'implémentation prévue du projet, on se base sur les modèles choisis et expliqués dans les artefacts tels le protocole de communication et le document d'architecture logiciel. Ces documents se basent sur les requis et les exigences du client.

Lors du développement des fonctionnalités, lorsqu'il y a des modifications au niveau de l'implémentation prévu pour celles-ci ou des requis dans ces documents, l'équipe discute de la modification et si celle-ci est valide. Au moment où le changement est accepté par l'équipe, les modifications sont apportées directement dans les artefacts. Les modifications aux artefacts sont gardées.

3.2. Contrôle de la qualité

Des tests unitaires sont rédigés pour toutes les fonctionnalités du projet. Ces tests permettent de tester les nouvelles fonctionnalités, mais surtout de s'assurer que du nouveau code ne brise pas des fonctionnalités déjà complétées.

Dès qu'un nouveau commit est fait sur une branche, un pipeline construit les clients et le serveur et effectue les suites de tests de ceux-ci. Avant d'entreprendre un merge, l'instance GitLab exécute à nouveau les tests sur le code résultant, s'assurant ainsi de ne pas introduire du code erroné dans le projet. Dans le cas où la pipeline échoue, la personne qui a poussé le code doit s'occuper d'arranger les tests en conséquence. Dans l'éventualité que c'est un problème d'intégration où nous avons besoin de l'aide d'un autre membre de l'équipe, nous avons une plateforme de communication d'équipe (Discord) qui nous permet de rester en contact.

Le plan de test nous permettra de spécifier différents types de tests qui nous permettront de s'assurer que notre application est fonctionnelle, stable, utilisable et performante.

Lorsque des actions correctives sont nécessaires pour des problèmes qui auraient passés l'assurance qualité, une rencontre d'équipe est organisée où l'on décide comment et qui s'occupe de la résolution du conflit.

3.3. Gestion de risque

La description des risques suit la convention suivante :

- Ampleur : sur une échelle de 1 à 10, 10 étant le risque le plus élevé. Cette analyse est basée sur la probabilité d'occurrence du risque, ainsi que ses impacts.
- Description : une description textuelle du risque ainsi que les problèmes attendus.
- Impact : échelle définissant la portée du risque
    - C – critique : 4 (affecte le projet en entier)
    - E – élevé : 3 (affecte les fonctionnalités principales du système)
    - M – moyen : 2 (devrait être maîtrisable en appliquant une stratégie d'atténuation adéquate)
    - F – faible : 1 (l'acceptation du risque est une stratégie envisageable)
- Facteurs : aspects (**métriques**) du système pouvant être compromis.
- Stratégie de gestion : mesures à prendre afin de gérer le risque.
- Probabilité d'occurrence :
    - Très peu probable: 1
    - Peu probable: 2

- Probable : 3
- Très probable : 4
- Extrêmement probable : 5

- L'ampleur, décrite plus haut, sera donc :
  - Ampleur = Probabilité d'occurrence * Impact que nous ramenons sur une échelle de 10 en divisant par deux en arrondissant à la hausse. Comme la valeur maximale d'ampleur est 5 * 4 = 20, alors nous divisons par 20 et multiplions par 10 ce qui est équivalent à diviser par 2.

<1> - Non disponibilité du serveur					
Ampleur	**Description**	**Impact**	**Probabilité d'occurrence**	**Facteurs**	**Stratégie de gestion**
4	La non disponibilité du serveur de l'application (il s'agit d'un serveur personnel et non Azure/AWS) dû à une panne de courant potentielle. - Connexion et utilisation de l'application impossible sur ordinateur et mobile	C (4)	Peu probable (2)	Disponibilité du serveur (uptime)	-L'installation d'un serveur de secours sur AWS/Azure ou d'un autre serveur personnel en parallèle.

<2> - Manque de planification de travail d'un membre					
Ampleur	**Description**	**Impact**	**Probabilité d'occurrence**	**Facteurs**	**Stratégie de gestion**
2	Le manque de planification du travail prévu pour chaque sprint pour chacun des membres. -Perte de temps lors des heures de travail -Membre ne sait pas travailler sur quelle partie de l'application	F (1)	Probable (3)	Temps de planification de la semaine de travail	-Avoir des rencontres plusieurs fois par semaine pour informer les autres membres du travail fait et prévu pour permettre une vue d'ensemble globale sur l'avancement du projet.

<3> - Mauvaise connection internet					
Ampleur	**Description**	**Impact**	**Probabilité d'occurrence**	**Facteurs**	**Stratégie de gestion**
4	L'utilisateur ne possède pas une connection internet assez stable pour permettre l'utilisation des services de l'application - Connexion et utilisation de l'application impossible sur ordinateur et mobile	C (4)	Peu probable (2)	Latence du système	-Nous n'y pouvons rien.

<4> - Problème de fusion du code (Merge)					
Ampleur	**Description**	**Impact**	**Probabilité d'occurrence**	**Facteurs**	**Stratégie de gestion**
4	Lors des “merge” de code sur les branches de git, il y a potentiellement des problèmes de compilation dû au nouveau modules ajoutés. -Programme ou fonctionnalités qui ne compile pas sur ordinateur et mobile	M (2)	Très probable (4)	Quantité de pipeline échoué lors de l’intégration du code	-Installer une infrastructure de test automatique qui déploie les tests après chaque merge. -L’utilisation de “code review” sur gitlab pour avoir l’accord de l’équipe pour les modifications du code source.

<5> - Fonctionnement de la librairie de traitement d’image					
Ampleur	**Description**	**Impact**	**Probabilité d'occurrence**	**Facteurs**	**Stratégie de gestion**
6	Il y a beaucoup d'incertitudes sur l’utilisation de la librairie de traitement d’image “potrace” proposée qui prend une image et la transforme en image vectorielle.	E (3)	Très probable (4)	Qualité de l’image produite	-Rechercher une librairie alternative -Dans l’éventualité qu’une librairie alternative n’existe pas : modifier nos exigences de la création de la paire mot-image.

<6> - Manque de cohésion des interfaces utilisateur des différents clients					
Ampleur	**Description**	**Impact**	**Probabilité d'occurrence**	**Facteurs**	**Stratégie de gestion**
5	Les interfaces utilisateur du client sur ordinateur et sur tablette peuvent ne pas être assez semblables et offrir une expérience d’utilisation différente.	M (2)	Extrêmement probable (5)	Cohésion des interfaces	-Utiliser un outil de schématisation d’interfaces, comme Figma pour produire une référence d’interface pour les différents clients.

<7> - Non respect de l’échéancier pour le client léger					
Ampleur	**Description**	**Impact**	**Probabilité d'occurrence**	**Facteurs**	**Stratégie de gestion**
4	Le manque de familiarité avec l’utilisation d’Android Studio ainsi qu’avec le langage de programmation Kotlin du client léger peut causer des délais donc un non-respect de l'échéancier pour celui-ci.	M (2)	Très probable (4)	Les fonctionnalités non implémentées à cause du délai.	-Mise en place de rencontres hebdomadaires SCRUM pour tenir l’équipe au courant et ajuster l'échéancier.

<8> - Attaque sur l'intégrité du système					
Ampleur	**Description**	**Impact**	**Probabilité d'occurrence**	**Facteurs**	**Stratégie de gestion**
1	Il y a la possibilité d'une attaque qui affecte l'intégrité du système ainsi que les bases de données.	F (1)	Très peu probable (1)	Les informations personnelles et de connexions des usagers	-Utiliser un système qui encrypte les données lors du transfert entre le client et le serveur.

<9> - Panne d'électricité affectant le serveur					
Ampleur	**Description**	**Impact**	**Probabilité d'occurrence**	**Facteurs**	**Stratégie de gestion**
4	Il y a la possibilité d'une panne d'électricité pour le bâtiment qui contient le serveur ce qui le rend non disponible.	C (4)	Peu probable (2)	Disponibilité du serveur (uptime)	-Installation d'une génératrice qui s'active automatiquement lorsqu'il y a une panne d'électricité et s'occupe de maintenir la disponibilité du serveur.

3.4. Gestion de configuration

Gestion des problèmes

- Dans le cas de problèmes en lien avec des changements liés aux tickets sur jira, il est consensus que la personne responsable du ticket est aussi responsable de la résolution du problème.
- Dans le cas de problèmes externes en lien avec des modifications de codes ou de "merges", il est demandé de créer un ticket dans la section en lien au problème sur jira.
- Tous les membres de l'équipe sont assignés à la résolution des tickets "problème", mais après chaque changement au code source, la personne qui effectue le changement doit aussi s'assurer qu'il n'y a pas d'introduction de problèmes.

Gestion des changements

- Pour la gestion de changements, mise en place d'un système de gestion des "merges" au code source par l'équipe sur gitlab. Il faut donc l'approbation d'au moins deux autres membres de l'équipe après un code review pour permettre de modifier la branche de code principale.
- Pour les changements qui sont sur des branches de développement, on a uniquement besoin de se conformer aux normes de nomenclature des commits.
- Chaque semaine, les mercredis à 12h ainsi que les samedis à 13h nous allons nous rencontrer pour un "scrum meeting". L'ordre du jour pour chaque scrum est de donner le tour de parole à chaque personne pour qu'elle présente son progrès depuis la dernière rencontre, qu'elle demande de l'aide si elle est bloquée sur une fonctionnalité et finalement ce qu'elle fera d'ici les prochains jours. Ainsi l'équipe sera à jour sur les changements apportés à l'application et tout le monde aura une meilleure vue d'ensemble du projet.

Normes de nommage

- Sur Jira, les tickets ont chacuns un tag (ex: GL3H21204-*) où l'étoile est le numéro du ticket. Nous créons nos branches sur gitlab à l'aide de ce numéro de ticket et nos commit se basent aussi sur celui-ci pour garder l'uniformité du code. Les sous-tâches sur Jira possèdent aussi un numéro. Voici un exemple :
  - P-13-compiler-electron-push-projet-2
  - P-13-22-intégrer-electron
  - P-13-21-strip-client-lourd

  Pour faire un commit sur le ticket 13-22 de l'intégration de électron, on pourrait faire comme :
  git commit -m "P-13-22 electron now works"

Gitlab s'occupe de refuser tout commit qui ne se conforme pas aux normes établies pour éviter des problèmes de compréhension plus tard au niveau des modifications de code.

- Pour les versions des artéfacts, nous allons simplement opter pour une incrémentation de la version. La première version sera 1.0, la version suivante sera 1.1 et ainsi de suite.

4. Échéancier du projet

Livrable	Lot	Heures-personnes	Date de début	Date de fin	Temps Total (Heures)
Sprint 1 (1 sem)			20 janvier	26 janvier	24
	Configuration initiale du Jira	6	20 janvier	26 janvier	
	Rédaction de la liste des exigences	12	20 janvier	26 janvier	
	Configuration du répertoire Gitlab et définition des normes	6	20 janvier	26 janvier	
Fin Sprint 1 : À la fin de ce sprint, nous devrions avoir notre infrastructure avec git et jira ainsi qu'un début de la rédaction de la liste des exigences.					
Sprint 2 (1 sem)			27 janvier	2 février	54
	Mise en place de la pipeline Gitlab	6	27 janvier	2 février	
	Dernières modifications à la liste des exigences	2	27 janvier	2 février	
	Rédaction du document SRS	18	27 janvier	2 février	
	Création du projet initial client léger	10	27 janvier	2 février	
	Conversion de la solution du client lourd pour Électron	10	27 janvier	2 février	
	Conversion de la solution du serveur	8	27 janvier	2 février	

Fin Sprint 2 : À la fin de ce sprint, nous devrions avoir fini la mise en place du pipeline sur gitlab pour le serveur ainsi qu'un début de rédaction du SRS, la fin de la liste d'exigences ainsi qu'une version fonctionnelle du projet de base (Projet 2) sur le client léger et le client lourd.					
Sprint 3 (2 sem)			3 février	16 février	130
	Dernières modifications au document SRS	6	3 février	16 février	
	Rédaction du document de plan de projet	20	3 février	16 février	
	Rédaction du document de protocole de communication	10	3 février	16 février	
	Rédaction du document d'architecture logicielle	24	3 février	16 février	
	Prototype communication client lourd	20	3 février	16 février	
	Prototype communication client léger	25	3 février	16 février	
	Prototype communication serveur	25	3 février	16 février	
Remise de la réponse à l'appel d'offre				19 février	
Fin Sprint 3 : À la fin de ce sprint, nous devrions avoir terminé le SRS, le plan de projet, le protocole de communication ainsi que le document d'architecture logiciel. Nous devrions avoir une version fonctionnelle du chat comme prototype de communication entre les clients et le serveur.					
Sprint 4 (1 sem)			17 février	23 février	85
	Création de compte, et gestion de compte. Client lourd.	25 heures	17 février	23 février	
	Création de compte, et gestion de	35 heures	17 février	23 février	

	compte. Client léger.				
	Début du système de liste d'amis côté serveur. Structure de la base de données, "endpoints" pour les demandes d'ajout et suppression d'amis.	25 heures.	17 février 17 février	23 février 23 février	
Fin Sprint 4 : À la fin de ce sprint, nous devrions avoir la création et gestion de compte de terminé pour les deux clients ainsi qu'un début des fonctionnalités de la liste d'amis du côté serveur.					
Sprint 5 (1 sem)			24 février	2 mars	120
	Menu principal avec options implémentés: déconnexion, créer lobby, rejoindre lobby. Client Lourd	24 heures	24 février	2 mars	
	Afficher la liste de lobby disponible et rejoindre lobby. Client Lourd	24 heures	24 février	2 mars	
	Menu principal avec options implémentés: déconnexion, créer lobby, rejoindre lobby. Client Léger	24 heures	24 février	2 mars	
	Gestion des commandes d'un dessin. Serveur	16 heures	24 février	2 mars	
	Mode de jeu classique. Serveur	32 heures	24 février	2 mars	
Fin Sprint 5 : À la fin de ce sprint, nous devrions avoir fini le menu principal pour les deux clients. Nous devrions être capable d'afficher les différents lobby qui se trouve sur le serveur et pouvoir les rejoindre. Finalement, avoir la gestion des commandes de dessins du serveur terminé ainsi qu'un début du mode de jeu classique pour le					

serveur.					
Sprint 6 (1 sem)			3 mars	9 mars	106
	Lié espace de dessin à serveur. Client Lourd	16 heures	3 mars	9 mars	
	Liste d'amis: ajouter ami, retirer ami. Client Lourd	6 heures	3 mars	9 mars	
	Début mode de jeu classique jouer réel seulement	12 heures	3 mars	9 mars	
	Espace de dessin: crayon, efface. Client Léger	16 heures	3 mars	9 mars	
	Liste d'amis: ajouter ami, retirer ami. Client Léger	12 heures	3 mars	9 mars	
	Comportement joueur virtuel. Serveur	36 heures	3 mars	9 mars	
	Réception et gestion de paire mot-image	8 heures	3 mars	9 mars	
Fin Sprint 6 : À la fin de ce sprint, nous devrions avoir une espace de dessin utilisable avec les outils, pouvoir utiliser les fonctionnalités de la liste d'amis et un début du mode de jeu classique pour les clients avec des joueurs réels. Aussi, avoir un début de gestion des joueurs virtuels du côté serveur.					
Sprint 7 (1 sem)			10 mars	16 mars	118
	Création paire mot-image manuelle 1. Client Lourd	8 heures	10 mars	16 mars	
	Comportement joueur virtuel simple. Client Lourd	16 heures	10 mars	16 mars	
	Mode de jeu classique avec joueur virtuelle	12 heures	10 mars	16 mars	

	Création paire mot-image manuelle 1. Client Léger	6 heures	10 mars	16 mars	
	Comportement joueur virtuel simple. Client Léger	16 heures	10 mars	16 mars	
	Mode de jeu classique avec joueur virtuel. Client Léger	12 heures	10 mars	16 mars	
	Mode de jeux solo et coop. Serveur	36 heures	10 mars	16 mars	
	Historique des utilisateurs Serveur	12 heures	10 mars	16 mars	
Fin Sprint 7 : À la fin de ce sprint, nous devrions avoir une création de paire mot-image fonctionnelle en grande partie, un comportement simple pour les joueurs virtuels dans les deux clients, la gestion des modes de jeux coop et solo du côté serveur ainsi que l'historique des utilisateurs.					
Sprint 8 (1 sem)			17 mars	23 mars	95
	Création paire mot-image Manuelle 2. Client Lourd	8 heures	17 mars	23 mars	
	Création paire mot-image Assistée 1. Client Lourd	8 heures	17 mars	23 mars	
	Mode de jeu Solo. Client Lourd	6 heures	17 mars	23 mars	
	Mode de jeu Coop. Client Lourd	6 heures	17 mars	23 mars	
	Création paire mot-image Manuelle 2. Client Léger	8 heures	17 mars	23 mars	
	Création paire mot-image Assistée 1.	7 heures	17 mars	23 mars	

	Client Léger				
	Mode de jeu Solo. Client Léger	6 heures	17 mars	23 mars	
	Mode de jeu Coop. Client Léger	4 heures	17 mars	23 mars	
	Gestion de canal de discussion privée. Serveur	12 heures	17 mars	23 mars	
	Historique des canaux de discussion. Serveur	18 heures	17 mars	23 mars	
	Personalité des joueurs virtuelles	12 heures	17 mars	23 mars	
Fin Sprint 8 : À la fin de ce sprint, nous devrions avoir complètement terminé la création des paires mot-image pour le client lourd, le mode de jeu coop et solo fonctionnel pour les deux clients, avoir accès aux canaux de discussion ainsi que la personnalité des joueurs virtuels implémentés.					
Sprint 9 (1 sem)			24 mars	30 mars	92
	Profil utilisateur et historique. Client Lourd	12 heures	24 mars	30 mars	
	Message privé à un ami. Client Lourd	8 heures	24 mars	30 mars	
	Historique des canaux de discussion. Client Lourd	16 heures	24 mars	30 mars	
	Personnalité des joueurs virtuels. Client Lourd	4 heures	24 mars	30 mars	
	Profil utilisateur et historique. Client Léger	12 heures	24 mars	30 mars	
	Message privé à un ami. Client Léger	8 heures	24 mars	30 mars	

	Historique des canaux de discussion. Client Léger	16 heures	24 mars	30 mars	
	Personnalité des joueurs virtuels. Client Léger	4 heures	24 mars	30 mars	
	Réinitialisation mot de passe. Serveur	12 heures	24 mars	30 mars	
Fin Sprint 9 : À la fin de ce sprint, nous devrions avoir le tableau de bord avec l'affichage de l'historique de connexions, de parties et les informations générales de terminé pour les deux clients. Les messages privés devraient être fonctionnels pour les clients ainsi que l'affichage des personnalités des joueurs virtuels dans les parties.					
Sprint 10 (1 sem)			31 mars	6 avril	60
	Création paire mot-image Assisté 2 Client Lourd	24 heures	31 mars	6 avril	
	Tutoriel Client Léger et Client Lourd	12 heures	31 mars	6 avril	
	Tampon. Serveur (fix divers et tests)	24 heures	31 mars	6 avril	
Fin Sprint 10 : À la fin de ce sprint, nous devrions avoir terminé le tutoriel pour les deux clients ainsi qu'une période tampon pour la correction de problèmes du côté serveur ainsi que de tests.					
Sprint 11 (1 sem)			7 avril	13 avril	52
	Création paire mot-image Assisté 3 Client Lourd	16 heures	7 avril	13 avril	
	Réinitialisation de mot de passe. Client Léger	12 heures	7 avril	13 avril	
	Tampon. Serveur (fix divers et tests)	24 heures	7 avril	13 avril	
Fin Sprint 11 : À la fin de ce sprint, nous devrions avoir terminé la création de paire mot-image assisté 3 pour le client lourd et la réinitialisation de mot de passe pour le client léger. Finalement, avoir une autre période tampon					

pour régler des problèmes de communication avec les clients et faire des tests.					
Sprint 12 (1 sem)			14 avril	18 avril	72
	Réinitialisation de mot de passe. Client Lourd	16 heures	14 avril	18 avril	
	Tampon. ClientLéger	32 heures	14 avril	18 avril	
	Tampon. Serveur	24 heures	14 avril	18 avril	
Remise du produit final				19 avril	
Fin Sprint 12 : À la fin de ce sprint, nous devrions avoir une application complètement fonctionnelle selon les requis et la révision des différents documents pour la remise finale, y compris le plan de test et résultat de test.					

5. Équipe de développement

Les membres de l'équipe sont les suivants:

- Étienne Plante
  - Bonne expérience en développement full-stack, soit l'intégration front et back-end.
  - Responsable du développement du client léger.

- Samuel Poulin
  - Possède un niveau de connaissances élevé en développement web et en back-end.
  - Responsable du développement du client lourd.

- Simon Malouin
  - Expertise en développement de front-end grâce à une longue expérience en industrie ainsi qu'une bonne compréhension des systèmes d'exploitation.
  - Responsable du développement du client lourd ainsi que de l'entretien du serveur.

- Antoine Morcel
  - Connaît bien le développement d'APIs et de back-end grâce à une expérience en industrie.
  - Responsable du développement du serveur.

- Guillaume Beausoleil
  - Bonnes connaissances en développement de front-end.
  - Responsable du développement du client léger.

- Maxime Fecteau
  - Connaissances particulières en développement de back-end sur le cloud grâce à une expérience en industrie.
  - Responsable du développement du serveur.

6. Entente contractuelle proposée

Nous proposons une entente contractuelle à livraison clé à main. Nous pensons que cette option semble la plus pratique pour les deux parties, puisque les requis sont considérablement clairs et l'échéancier est déjà établi. De plus, de cette manière le promoteur a une implication minimale tout au long du projet. Étant donné la liste des exigences déjà bien établies, le risque de devoir négocier un changement de spécification est bas ainsi ce type de contrat semble pertinent. Notre équipe de 6 développeurs travaillera de manière à ce que l'équipe au complet s'occupe de la gestion de projet ainsi, nous chargerons l'équivalent d'un gestionnaire de projet et 5 développeurs. Ainsi, nous arrivons à un produit clé en main pour une somme totale de 105 000$ selon les taux horaires proposés dans l'appel d'offres.

1008 heures / personne donne 168 heures par personne

168 heures * 5 * 100$/h par développeurs = 84 000$

168 heures * 125$/heures pour un gestionnaire de projet = 21 000$

La somme totale sera donc 105 000$