Équipe 204

Projet d’évolution d’un logiciel
Plan de projet

Version 1.0

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

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

Afin de réaliser un jeu de dessin multijoueur 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. Une 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.

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 gardés.

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.

Lorsque du nouveau code aboutit sur la branche develop ou master, des artéfacts (exécutables) sont produits afin de pouvoir être utilisés à l'interne ou pour en faire la démonstration au client.

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 (affecte le projet en entier)
    - E – élevé (affecte les fonctionnalités principales du système)
    - M – moyen (devrait être maîtrisable en appliquant une stratégie d'atténuation adéquate)
    - F – faible (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.

Ampleur = (Probabilité + impact)/10

<1> - Non disponibilité du serveur				
Ampleur	**Description**	**Impact**	**Facteurs**	**Stratégie de gestion**
7	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	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**	**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	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**	**Facteurs**	**Stratégie de gestion**
5	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	E	Latence du système	-Installation d'un service internet stable et sécurisé par l'utilisateur,

<4> - Problème de fusion du code (Merge)				
Ampleur	**Description**	**Impact**	**Facteurs**	**Stratégie de gestion**
5	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	E	Quantité de "merge" dans le code source	-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.

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.

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.

4. Échéancier du projet

Livrable	Lot	Heures-personnes	Date de début	Date de fin
Sprint 1 (1 sem)			20 janvier	26 janvier
	Configuration initiale du Jira	6		
	Rédaction de la liste des exigences	6		
	Configuration du répertoire Gitlab et définition des normes	6		
Sprint 2 (1 sem)			27 janvier	2 février
	Mise en place de la pipeline Gitlab	6		
	Dernières modifications à la liste des exigences	2		
	Rédaction du document SRS	16		
	Création du projet initial client léger	10		

	Conversion de la solution du client lourd pour Électron	10		
	Conversion de la solution du serveur	8		
Sprint 3 (2 sem)			3 février	16 février
	Dernières modifications au document SRS	6		
	Rédaction du document de plan de projet	20		
	Rédaction du document de protocole de communication	8		
	Rédaction du document d'architecture logicielle	25		
	Prototype communication client lourd	20		
	Prototype communication client léger	25		
	Prototype communication serveur	25		
Remise de la réponse à l'appel d'offre				19 février
Sprint 4 (1 sem)			17 février	23 février
	Création de compte, et gestion de compte. Client lourd.	25 heures		
	Création de compte, et gestion de compte. Client léger.	35 heures		

	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.	21 heures.		
Sprint 5 (1 sem)			24 février	2 mars
	Menu principal avec options implémentés: déconnexion, créer lobby, rejoindre lobby. Client Lourd	12 heures		
	Afficher la liste de lobby disponible et rejoindre lobby. Client Lourd	12 heures		
	Menu principal avec options implémentés: déconnexion, créer lobby, rejoindre lobby. Client Léger	12 heures		
	Menu principal avec options implémentés: déconnexion, créer lobby, rejoindre lobby. Client Léger	12 heures		
	Gestion des commandes d'un dessin. Serveur	16 heures		
	Mode de jeu classique. Serveur	8 heures		
Sprint 6 (1 sem)			3 mars	9 mars
	Lié espace de dessin à serveur. Client Lourd	8 heures		
	Liste d'amis: ajouter ami, retirer ami. Client Lourd	6 heures		

	Début mode de jeu classique jouer réel seulement	10 heures		
	Espace de dessin: crayon, efface. Client Léger	16 heures		
	Liste d'amis: ajouter ami, retirer ami. Client Léger	8 heures		
	Comportement joueur virtuel. Serveur	8 heures		
	Réception et gestion de paire mot-image	8 heures		
Sprint 7 (1 sem)			10 mars	16 mars
	Création paire mot-image manuelle 1. Client Lourd	8 heures		
	Comportement joueur virtuel simple. Client Lourd	8 heures		
	Mode de jeu classique avec joueur virtuelle	8 heures		
	Création paire mot-image manuelle 1. Client Léger	6 heures		
	Comportement joueur virtuel simple. Client Léger	9 heures		
	Mode de jeu classique avec joueur virtuel. Client Léger	9 heures		
	Mode de jeux solo et coop. Serveur	12 heures		
	Historique des	12 heures		

	utilisateurs Serveur			
Sprint 8 (1 sem)			17 mars	23 mars
	Création paire mot-image Manuelle 2. Client Lourd	7 heures		
	Création paire mot-image Assistée 1. Client Lourd	7 heures		
	Mode de jeu Solo. Client Lourd	6 heures		
	Mode de jeu Coop. Client Lourd	4 heures		
	Création paire mot-image Manuelle 2. Client Léger	7 heures		
	Création paire mot-image Assistée 1. Client Léger	7 heures		
	Mode de jeu Solo. Client Léger	6 heures		
	Mode de jeu Coop. Client Léger	4 heures		
	Gestion de canal de discussion privée. Serveur	12 heures		
	Historique des canaux de discussion. Serveur	6 heures		
	Personalité des joueurs virtuelles	6 heures		
Sprint 9 (1 sem)			24 mars	30 mars
	Profil utilisateur et historique. Client Léger	6 heures		

	Message privé à un ami. Client Léger	8 heures		
	Historique des canaux de discussion. Client Léger	6 heures		
	Personnalité des joueurs virtuels. Client Léger	4 heures		
	Profil utilisateur et historique. Client Léger	6 heures		
	Message privé à un ami. Client Léger	8 heures		
	Historique des canaux de discussion. Client Léger	6 heures		
	Personnalité des joueurs virtuels. Client Léger	4 heures		
	Réinitialisation mot de passe. Serveur	12 heures		
Sprint 10 (1 sem)			31 mars	6 avril
	Création paire mot-image Assisté 2 Client Lourd	16 heures		
	Tutoriel Client Léger	8 heures		
	Tampon. Serveur			
Sprint 11 (1 sem)			7 avril	13 avril
	Création paire mot-image Assisté 3 Client Lourd	16 heures		
	Réinitialisation de mot de passe. Client Léger	6 heures		

	Tampon. Serveur			
Sprint 12 (1 sem)			14 avril	18 avril
	Réinitialisation de mot de passe. Client Lourd	4 heures		
	Tampon. ClientLéger			
	Tampon. Serveur			
Remise du produit final				19 avril

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 675 000$ selon les taux horaires proposés dans l'appel d'offres.