Guide de recommandation

Projet : Marketplace de la cryptomonnaie

Client : GitmeMoney

Table des matières

1 Objet du document

Le document suivant présente les recommandations à destination des parties prenantes du projet, et aussi les développeurs pour le projet de Marketplace de la cryptomonnaie.

Ce document n'est pas figé dans le temps, il pourra être amélioré par tous les membres de l'équipe de développement.

2 Les phases du projet

Le projet est constitué de plusieurs phases (process) comme décrit dans le sprint backlogue.

1. POC *Proof of concept*
2. EVA *Evaluation*
3. FEI *Frontend implementation*
4. BEI *Backend implementation*
5. AEI *API endpoint implementation*
6. CON *Container*
7. DMI *Deployement and migration*
8. OLI *ONLINE*

Chaque process possède quatre statues :

- TODO
- ENCOURS
- QA
- DONE

Le projet technique est découpé en quatre SBB :

- ACR *Solution d'achat de cryptomonnaie*
- VCR *Solution de vente de cryptomonnaie*
- EXA *Solution d'exposition d'API*
- BDD *Solution de stockage de données*

La navigation dans le product backlog se fait à partir de la phase ou le projet de situe.

Un espace est prévu dans chaque colonne de la phase (SBB) afin d'y ajouter les tickets associés.

Le backlog est pour le moment prérempli avec les grandes jalons du projet. Il est possible qu'en cours de projet des tickets soient ajoutés afin de répondre à des exigences de correction de bug, mise à niveau de dépendance…

Par exemple, lors de la phase de backend implementation, il est possible de créer des tickets afin de relier le front et le back de la colonne Vendre des cryptomonnaies.

3 Méthodes de travail

La technologie est souvent coûteuse, il a été constaté que près de la moitié des projets informatiques dépassent le budget alloué. Il est donc important d'étudier, gérer et prévoir les coûts futurs associés aux projets informatiques.

Les chefs de projets informatiques sont également confrontés à d'autres défis :

- L'évolution permanente de la technologie rend les projets informatiques existants obsolètes
- Un manque de communication et de vision commune entre les responsables informatiques et les dirigeants des entreprises
- Des contretemps et des courbes d'apprentissage abruptes lorsque les membres de l'équipe apprennent à utiliser les nouvelles technologies

De plus, les projets informatiques comportent généralement davantage de dépendances que les autres projets au sein d'une entreprise. Ils sont plus complexes et plus difficiles à suivre, et le moindre contretemps risque de provoquer l'interruption brutale de l'ensemble du projet.

De nos jours, le gaspillage d'argent a chuté de 20 % depuis que les entreprises se sont mises à gérer les projets informatiques différemment des autres processus.

Modèle en cascade	Méthode agile
La méthode de gestion de projet informatique la plus couramment utilisée est le modèle en cascade. Le but est d'effectuer le travail une étape après l'autre. Il s'agit d'une approche linéaire, c'est pourquoi elle fonctionne bien avec les diagrammes de Gantt, dans lesquels les tâches sont assignées selon un calendrier. Bien qu'idéal pour les projets présentant des interdépendances complexes, le modèle en	La méthode agile est en train de devenir l'une des méthodes de gestion de projet les plus populaires. D'abord utilisée dans le développement de logiciels, l'agilité s'est imposée dans tous les secteurs car l'accent est mis sur l'amélioration continue. Contrairement au modèle en cascade, où les projets peuvent être planifiés annuellement, l'agilité se fait en sprints courts (ou par lots).

cascade n'est pas aussi adaptable et réactif au changement que l'approche agile.	L'un des principaux avantages de cette approche est son adaptabilité. Le travail itératif équivaut à des boucles de rétroaction plus courtes, ce qui signifie qu'il est plus facile d'adapter les produits à la technologie dominante et/ou aux préférences des clients. Les projets agiles n'étant généralement pas planifiés sur de longues périodes, de nombreux chefs de projets informatiques gèrent les tâches via des tableaux Kanban plutôt qu'avec un diagramme de Gantt.

3.1 Usage d'une méthode hybride

La méthode hybride combine des éléments du modèle en cascade et de la méthode agile. Cette méthode allie équipes agiles et projets par étapes. Les entreprises peuvent utiliser la planification à long terme pour les objectifs stratégiques, tandis que les projets informatiques sont réalisés en sprints en appliquant la méthode agile.

3.2 Avancement de la Roadmap

Le découpage du sprint backlog est réalisé en fonction de la roadmap et du SBB fournie pour ce projet.

Comme pour la méthode Scrum, l'avancé de la roadmap se fait sous la forme de sprint de **4 semaines**.

Ainsi, des réunions régulières permettront de noter les avancés du projet au travers des process (POC, évaluation, implémentations…).

3.3 Organisation des rencontres entre les parties prenantes

3.3.1 Réunion de lancement

La réunion de lancement pour un projet (ou kickoff meeting) est la première rencontre entre les intervenants du commanditaire du projet et l'équipe projet (qui sera en charge de mener à bien le projet considéré). Elle a pour objectif de présenter les membres de l'équipe projet aux intervenants du commanditaire et de clarifier le rôle de chacun. D'autres éléments-clés du projet impliquant le commanditaire peuvent également faire partie de l'ordre du jour de cette réunion (comme le calendrier, la périodicité et le format des rapports d'avancement, etc.).

Objectifs :

- Présentation de l'équipe

- Présentation de la roadmap
- Présentation du product backlog
- Présentation du planning d'avancement

3.3.2 Sprint

Un Sprint est une itération. Il dure une période de 4 semaines pendant laquelle une version terminée et utilisable du produit est réalisée. Un nouveau Sprint commence dès la fin du précédent. Chaque Sprint implique un objectif et une liste de fonctionnalités à réaliser.

3.3.3 Planification d'un Sprint

Les tâches à accomplir pendant le Sprint sont déterminées par l'ensemble de l'équipe Scrum lors du meeting de planification du Sprint. Celle-ci se limite à 8 heures pour les Sprints d'un mois. Elle permet à l'équipe d'établir les problématiques qu'elle va traiter au cours de ce Sprint.

3.3.4 Revue du Sprint

Il s'agit du bilan du Sprint réalisé. Une fois le Sprint terminé, l'équipe Scrum et les parties prenantes se réunissent pour valider ce qui a été accompli. Cette réunion dure 4 heures maximum.

3.3.5 Rétrospective du Sprint

La rétrospective est interne à l'équipe Scrum et dure 3 heures pour un Sprint d'un mois. Elle vise l'adaptation aux changements et l'amélioration continue du processus de réalisation. L'équipe se sert de la rétrospective pour passer en revue le Sprint terminé et déterminer ce qui a bien fonctionné et ce qu'il faut améliorer.

3.3.6 Mêlée quotidienne

Cette réunion journalière de 15 minutes est très importante. Elle se fait debout, afin d'éviter de s'éterniser. C'est ce qui explique que l'on parle de "stand-up meeting". Ce meeting a pour but de faire un point sur la progression quotidienne du Sprint. Cette rencontre permet à l'équipe de synchroniser ses activités et de faire un plan pour les prochaines 24 heures.

La mêlée a lieu à la même heure et au même endroit chaque jour. Chaque membre de l'équipe de développement doit répondre à ces trois questions :

- Qu'est-ce qu'il a réalisé la veille ?

- Qu'est-ce qu'il va accomplir aujourd'hui ?
- Quels sont les obstacles qui le retardent ?

3.3.7 La réunion de travail : pour approfondir une problématique

L'objectif de la réunion de travail est de valider le plan d'actions pour résoudre les problèmes. Elle réunit la maîtrise d'œuvre, les responsables de sous-projet, les contributeurs concernés et éventuellement le sponsor du projet. Cette réunion est organisée en fonction des besoins et de l'avancement du projet. Les différentes réunions de travail correspondent :

- aux réunions de préparation avant le kick off ;
- aux réunions pour approfondir une problématique après le kick off pour préparer les points d'avancement.

3.3.8 La réunion de clôture COPIL

La réunion de clôture du COPIL permet un retour d'expérience entre la MOA, la MOE et le CODIR. Elle est organisée une seule fois à la fin du projet.

- Les documents à prévoir pour la réunion de clôture du COPIL : le support de retour d'expérience

3.3.9 La réunion de clôture

Organisée après la réunion du COPIL, la réunion de clôture permet un retour d'expérience et une reconnaissance du travail effectué par les différents contributeurs. Elle est organisée une seule fois en présence de la maîtrise d'œuvre.

- Les documents à prévoir pour la réunion de clôture : le support de retour d'expérience

3.4 Les outils utiles à l'agilité

3.4.1 Le product backlog ou carnet du produit

Il s'agit d'une liste hiérarchisée des exigences initiales du client concernant le produit à réaliser. Ce document évolue sans cesse durant le projet, en fonction des besoins du client. Le product owner est responsable du product backlog.

3.4.2 Le Sprint backlog ou carnet de Sprint

C'est le plan détaillé de la réalisation de l'objectif du Sprint, défini lors de la réunion de planification du Sprint. Le Sprint backlog est mis à jour régulièrement par l'équipe, afin

d'avoir une vision précise de la progression du Sprint.

3.4.3 L'incrément Produit

Il s'agit de l'ensemble des éléments terminés du product backlog pour le Sprint en cours, ainsi que ceux des Sprints précédents. L'incrément doit fonctionner et être utilisable.

3.4.4 Le Burndown Chart ou graphique d'avancement

Ce graphique simple indique l'état d'avancement dans la réalisation des tâches du Sprint backlog. Il s'agit du tracé de la charge de travail restante – généralement exprimée en heures – en fonction du temps, exprimé en jours. Le Burndown Chart est actualisé tous les jours par le Scrum Master après le meeting Scrum.

4 Bonne pratique et standards de développement

4.1 Utilisation de règles de codage

Les principales raisons qui font que l'utilisation de règles de codage est essentiel lors du développement d'un projet :

- Une meilleure lisibilité du code. En effet, appliquer ces règles permet d'écrire, et donc de pouvoir lire, du code plus facilement. C'est encore plus vrai pour le travail en équipe ; il faut écrire du code propre pour que les collaborateurs puissent se relire. Aucun développeur n'aime récupérer du code mal écrit.
- Ce qui découle de cette première raison, c'est une meilleure maintenabilité. Un code plus lisible, plus commenté, est bien plus facile à modifier, en cas de bugs ou d'ajout de nouvelles features. C'est un gain de temps et d'énergie.
- Un respect des spécificités du langage utilisé. Lorsque des bonnes pratiques sont établies, elles doivent respecter les spécificités du langage avec lequel on code (ici JAVA). Donc, suivre ces bonnes pratiques permettra de moins faire d'erreurs lors de l'écriture du dit code.
- Une meilleure **gestion des performances**. Les conventions de codage devraient également être pensées de manière à mieux gérer les ressources physiques d'une machine, pour optimiser le produit, que ça soit au niveau du temps de chargement que des ressources demandées.

4.1.1 L'organisation des fichiers et l'arborescence du projet

Il est important, dès le commencement d'un projet, de bien réfléchir et définir une arborescence propre. C'est à dire, une organisation des fichiers qui soit adaptée au projet, au langage et/ou aux frameworks utilisés.

Typiquement, lorsqu'on débute dans le développement, on créé souvent tous les fichiers dans le même répertoire. Or, il est important de bien séparer les fichiers qui ont des utilités différentes.

Plus concrètement, et bien que cela dépende du langage et des outils, on pourrait voir créés des dossiers du genre :

- services : pour les fichiers contenant du code générique pouvant être utilisé ailleurs ;
- models/views/controllers : pour les projets utilisant le pattern MVC ;
- components/views : pour les projets utilisant un framework avec une approche par composant ;
- styles : pour stocker des fichiers de style type CSS.

Cette liste n'est pas complète, on peut y ajouter par exemple un dossier images, fonts, etc. Le but est ici d'éviter d'avoir des dizaines de fichiers dans le même répertoire.

4.1.2 Le nommage des fichiers

De même, il convient de nommer correctement ces fichiers. Souvent, cette règle dépend des entreprises et des conventions définies dans celles-ci. Mais, un des principes qu'on retrouve, c'est de nommer ces fichiers du même nom qu'on y nomme ce qui est défini.

Par exemple, pour une classe Automobile, on nommera le fichier Autmobile.extension.

Pour un service NetworkService, on appellera le fichier NetworkService.extension.

C'est une règle simple mais importante, car elle permet de comprendre l'utilité d'un fichier rien qu'en regardant son nom.

4.1.3 Le nommage des classes, variables, fonctions, etc.

Si on doit faire attention à la façon dont on nomme ces fichiers, il est encore plus important de faire attention à la façon dont on nomme ce qui s'y trouve : classes, fonctions, variables, etc.

Encore une fois, si ces règles vont changer d'une entreprise à l'autre, voire d'une équipe à l'autre, il y a certaines conventions qui restent plus ou moins invariables.

Déjà, il faut que le nom de ces éléments aient du sens. Éviter d'appeler un objet contenant les données d'un utilisateur myObject. user (si c'est une instance) ou userData aurait plus de sens.

Pour rentrer un peu plus dans les détails, on retrouve souvent ces règles dans le monde du codage :

- Pour une classe, on utilise du camelCase en commençant avec une majuscule : User ;
- Pour une fonction, on utilise du camelCase en commençant avec une minuscule : connectUser ;
- Idem pour une variable : connectedUser ;
- Une constante devrait s'écrire uniquement en majuscule : USER_ID.

4.1.4 L'indentation du code

L'indentation consiste à ajouter des tabulations ou des espaces à ses lignes de code ; c'est une autre composante importante des règles de codage. Cette convention, bien qu'unanime, est parfois un sujet de discorde entre les développeurs.

Certains utilisent les tabulations ; d'autres des espaces. Certains préfèrent utiliser deux espaces ; d'autres quatre.

Ces règles dépendent donc de chaque développeur, équipe ou entreprise. Mais quelle qu'elle soit, avoir un code bien indenté est essentiel à sa bonne lecture et maintenabilité.

4.1.5 Les commentaires

Souvent oubliés par les développeurs débutants, les commentaires ont pourtant une importance primordiale. Bien qu'ils n'aient aucune valeur ajoutée au niveau de l'exécution du code – d'où leur oubli – les commentaires sont essentiels à la bonne compréhension d'un morceau de code.

Sans toutefois tomber dans l'excès et commenter chaque ligne de code, il est important de mettre des commentaires pour chaque classe, fonctions, et blocs un peu complexes.

Ça sera utile non seulement pour les collaborateurs regarderont le code, mais aussi pour pour le développeur lui-même, quand il voudra reprendre son code après plusieurs semaines sans avoir travaillé dessus.

4.1.6 Taille des fichiers, fonctions

Ici, on va plus parler d'optimisation que que de vraie convention, mais la taille (en

nombre de lignes) d'un fichier et d'une fonction ont également une importance.

Lorsqu'on est développeur, on n'aime pas ouvrir un fichier JavaScript de 3000 lignes ni parcourir une fonction de 400 lignes. Tout simplement parce que comprendre ce que fait cette fonction requiert de la lire entièrement. Et lire 400 lignes de code sans perdre le fil, ce n'est pas simple !

Comment faire pour les réduire ? En ce qui concerne les fonctions, il est bon, lorsqu'elles deviennent trop imposantes, de les séparer en plusieurs fonctions. Il est facile pour un développeur de trouver facilement des redondances, ou des morceaux de code exécutant des fonctions annexes, et donc bons à exporter dans de nouvelles méthodes.

En ce qui concerne les fichiers, c'est encore plus simple. Si un fichier possède 40 ou 50 fonctions, il faut en externaliser certaines. On trouvera facilement des points communs entre certaines méthodes, et on pourra ainsi créer des groupes de fonctions qu'on exportera dans de nouveaux contrôleurs, services, etc.

4.1.7 Les parenthèses, point-virgules, etc.

La dernière bonne pratique que nous allons aborder ici, c'est celle qui concerne l'utilisation des parenthèses, des point-virgules, sauts de ligne, etc.

Cette règle, ou cet ensemble de règles, est souvent complètement dépendante d'une entreprise, et en particulier d'un CTO.

Par exemple, l'utilisation d'espace avant et après une parenthèse, et le saut à la ligne à la fin d'une condition if. Cela est totalement arbitraire.

Ou encore, l'utilisation ou non des ';' dans les langages où ils ne sont pas obligatoires, comme JavaScript ou Python. C'est encore une fois arbitraire.

4.2 Outil d'analyse de code

Il existe, qu'ils soient intégrés ou installés à travers des plugins, des outils de lecture et d'analyse de code. Par exemple, un outil de ce genre bien connu des développeurs web, c'est ESLint.

> Sous Visual Studio Code, ESLint est un plugin configurable, qui permet de définir des conventions, des règles, de codage. Si on ne respecte pas ces règles, des warnings apparaitront dans l'éditeur, voire pire, ils sera impossible de « commit » les modifications sans les avoir corrigés… C'est le moyen qu'ont trouvé certains CTO pour être sûrs que les conventions soient respectées.

Via ce type d'outils, il est également possible de formater automatiquement le code à

chaque sauvegarde.

4.3 Peer-programming

En jouant le rôle du driver ou du navigateur, le pair-programming est une très bonne pratique pour améliorer la qualité des développements et diffuser les connaissances au sein d'une équipe.

4.4 Code review

Plus classique mais presque aussi efficace, le respect des conventions de codage peut aussi être vérifié via des revues de code, notamment par des développeur plus expérimenté, ou directement avec le CTO.

Ces revues de code, qu'elles soient effectuées ensemble ou au travers de pull requests, permettent de faire comprendre aux développeurs en quoi leur code peut être amélioré, et en quoi ils ne respectent pas les conventions établies.

Il est possible que cette méthode ne soit pas agréables pour le développeur débutant, ces revues sont pourtant importantes dans une équipe de développement. Elles permettent également au développeur débutant de se perfectionner.

Pour aller plus loin : 50-conseils-pratiques-developpeurs