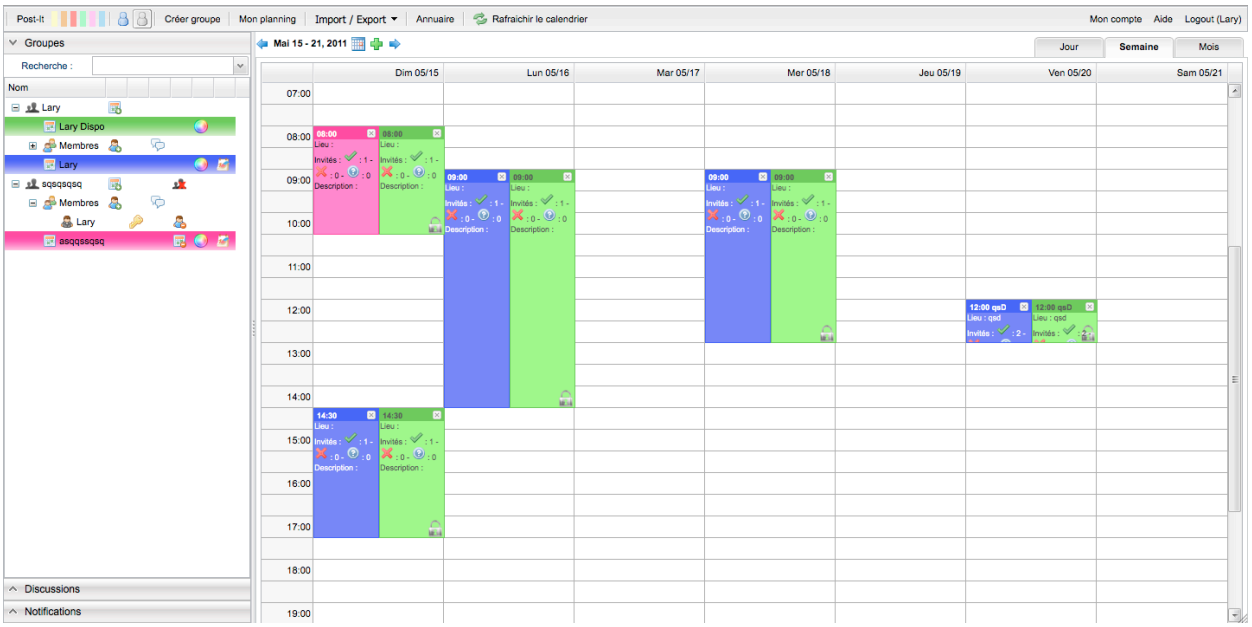
Calendrier Collaboratif :
Plan qualité

0. Introduction

Tout au long du développement d'une solution logicielle, il est important de maintenir une démarche rigoureuse, de l'analyse du besoin à la livraison. À cet effet, nous avons mis en place de multiples règles permettant de contraindre les méthodes de travail, dans le but de converger vers une qualité du logiciel maximale, c'est à dire de garantir l'aptitude du logiciel à satisfaire le besoin du client. Tous au long du déroulement du projet, ces règles ont fait l'objet d'améliorations continues.

La partie ci-dessous définit les concepts généraux de la méthode de fonctionnement que nous avons adopté, et le reste du document la façon dont nous l'avons mis en place.



Pierre Guillot - Benjamin Apprederisse - Lary Ciminera - Gwenaël Gevet - Rodolphe Lepigre

1. Règles générale de travail en équipe

0. Méthode de fonctionnement, génie logiciel

Notre méthode de fonctionnement est inspirée de l'extreme programming (XP). C'est une méthode agile de gestion de projet informatique adapté aux petits groupes. Le but est de rendre le projet le plus ouvert au changement et le plus flexible possible, à travers l'introduction de valeurs de base, de "principes" à respecter et de "bonnes pratiques".

L'idée de l'extreme programming est de pousser à l'extrême les pratiques simples :

- Puisque la revue de code est toujours bénéfique (pour réduire les bugs et améliorer les compétences des développeurs), elle sera faite en permanence (par un binôme, ou par un testeur)
- Puisque les tests sont fondamentaux, ils seront effectués avant chaque implémentation
- Puisque la conception est importante, elle sera faite tout au long du projet (refactoring)
- Puisque l'intégration des modifications est cruciale, elle sera toujours effectuée au plus tôt (idéalement quotidiennement)
- Pour pouvoir s'adapter rapidement aux changements, les cycles de développement sont très courts.

Les cycles de développement sont rapides (de quelques jours à une semaine) et simple :

- Définition du scénario client de l'itération courante
- Transformation du scénario en tâche à réaliser et en test fonctionnel
- Chaque développeur s'attribue une/des tâche(s)
- Lorsque tous les tests fonctionnels passent, on valide le scénario

Valeurs :

- La communication : elle doit être constante et intense. Elle permet d'éviter beaucoup de problèmes.
- La simplicité : la façon la plus simple d'arriver au résultat est la meilleure.
- Le feedback : Le retour d'information est primordial pour le programmeur. Les tests fonctionnels donnent l'avancement du projet. Les livraisons fréquentes permettent de tester les fonctionnalités rapidement.
- Le courage : Parfois, un nouveau scénario utilisateur implique des modifications majeures du code (voir de l'architecture même du logiciel). Une bonne dose de courage permet de faire face à ces situations. Les principes précédents (simplicité, feedback et communication) rendent cette tâche plus facile.
- Le respect.

2. Attitude et règles comportementales

Attitude générale à adopter par l'ensemble des personnes travaillant sur le projet :

Responsabilité personnelle :

- développement personnel
- santé physique et mentale
- équilibre travail – vie privée
- ponctualité

Respect des autres :

- respect des valeurs des autres et de leurs personnalités
- respect du travail des autres
- écoute et respect des points de vue différents du sien

Engagement :

- compréhension et acceptation des différences et de la fragilité de chacun
- dialogue et échange
- implication dans son travail et dans le projet de l'université
- aide et soutien aux autres personnes

Attitude du chef de projet vis à vis des collaborateurs :

- Empathie
- Discernement
- Éthique personnelle
- Curiosité et créativité
- Compréhension, bienveillance
- Prise en compte des particularités de chacun des membres de l'équipe
- Acceptation du droit à la fragilité, du droit à l'erreur et à l'imperfection
- Veille à l'épanouissement de chacun des membres de l'équipe
- Encouragement, reconnaissance vis à vis du travail accompli
- Égalité de traitement
- Organisation de l'aide et du soutien entre les développeurs
- Transparence des objectifs, des risques et des résultats
- Allocation des tâches clairement définies en fonction des capacités et des aspirations de chacun
- Contenu enrichissant du travail : apprentissage, acquisition de nouveaux savoirs
- Créer et maintenir un environnement et des conditions de travail de qualité
- Confiance en son équipe, encouragement de l'autonomie et de l'initiative
- Encouragement à la coopération, valorisation de la participation et du travail d'équipe
- Recherche du consensus dans la prise de décision

3. Responsabilités, fiches de poste

Nom	Rôle	Spécialité	équipe / binôme
Rodolphe Lepigre	Responsable technique	Gestion serveur, gestion serveur SVN, documents, post-it, color-picker.	jour
Benjamin Apprederisse	Développeur principal	Chat, calendrier, évènements, tâches, gestion des droits, des groupes, gestion interne BDD, notifications évènements.	nuit
Pierre Guillot	Chef de projet	Formulaire, modélisation BDD, rédaction des rapports, ergonomie, tests.	soir/matin
Lary Ciminera	Infographiste	Gestion des groupes, gestion interne BDD, notifications groupe, mise en place de structure pour l'interface des arbres.	nuit
Gwenael Gevet	Responsable des tests	Annuaire, import / export.	jour

4. Autonomie

Les créneaux à l'emploi du temps de l'université étant limités, la majorité du travail est réalisé hors université, à domicile. Les développeurs doivent donc être très autonomes, et beaucoup communiquer avec le reste de l'équipe. Le travail à réaliser par chaque programmeur est défini soit lors d'une réunion de projet, soit sur demande expresse d'un coéquipier ou du chef de projet.

Lorsqu'un module est terminé et fonctionnel il doit être publié sur le serveur SVN (gestionnaire de version), et un e-mail explicatif doit être envoyé aux membres de l'équipe. A chaque fois, les développeurs avertis vont lancer un jeu de tests sur leurs machines, avec leurs OS. Cette technique permet d'améliorer la fiabilité et la portabilité du code. Aussi, en cas de bug ou de problème, le développeur du module a encore en tête les détails de ce qu'il vient de faire et peu aisément remédier aux problèmes.
Il est demandé aux programmeurs d'être disponibles sur la messagerie instantanée de Gmail dès qu'ils travaillent, et d'être très réactifs aux e-mails.

2. Coordination

0. Google documents

La rédaction des documents livrables et des divers comptes-rendus de réunions est réalisée sur Google Documents. L'intérêt de ce support est de permettre l'édition collaborative en temps réelle. De plus, les documents sont dès leurs créations accessibles à tous à partir d'un navigateur, et ce de n'importe où, que ce soit sur le lieu de travail qu'à domicile, et même lors d'un déplacement. Ceci nous permet de nous détacher d'un logiciel de traitement de texte particulier sur un OS.

Règles applicables dans tous les Google docs partagés :
- Les indexes du sommaire commencent à 0. (On est des informaticiens quand même)
- Utiliser la fonction "commentaire" plutôt que d'écrire dans le corps du document
- Choisir une couleur par personne, et toujours écrire de cette couleur. C'est le chef de projet qui, lors de la relecture, valide les passages en les passant en noir.

Pierre écrit de cette couleur.
Benjamin écrit de cette couleur.
Rodolphe écrit de cette couleur.
Lary écrit de cette couleur.
Gwenaël écrit de cette couleur.

1. Réunions

Les objectifs de chaque réunion sont :
- Faire un point sur l'avancement global du projet
- Résoudre les problèmes techniques
- Prendre les décisions importantes
- Amélioration continue de la gestion du projet
- Amélioration continue du plan qualité

Les réunions ont lieu à chaque fois qu'un créneau "Projet" est marqué à l'emploi du temps de l'université (Dans le cas où il y a deux créneaux par jour, une seule réunion le matin).
Déroulement d'une réunion : tour à tour, chaque développeur explique où il en est dans son travail, les difficultés qu'il rencontre (idée inspirée du "Daily scrum meeting").
A chaque fois qu'une décision importante est prise, le chef de projet note dans un compte-rendu de réunion daté les détails de cette décision. (Souvent avec des photos prises du tableau où ont été débattus les points de vue). Le chef de projet rend librement accessible les comptes-rendus de réunion aux membres du projet sur google docs.

2. Échanges informels

Bilatéral

Il peut arriver qu'un programmeur ait besoin (ponctuellement) de l'assistance d'un collègue.
Dans l'ordre, les moyens de communication à privilégier sont :
- Chat Gmail
- E-mail perso
- Sms / téléphone
- Lors d'une réunion (à la fin)

Listes de diffusion

Après la publication d'un module ou d'une partie de module sur le serveur SVN, il est demandé au programmeur d'utiliser la liste de diffusion pour donner des précisions sur ce qu'il a fait (en plus de bien commenter sa publication de code source).
La liste de diffusion peut aussi être utilisée pour soumettre une nouvelle idée, ou un bug important. Il ne faut utiliser la liste de diffusion que si on veut un retour d'avis rapide sur un sujet. Sinon, il faut simplement ajouter l'idée ou le bug à la liste TODO courante.
Le chef de projet est chargé de réunir les points intéressants pour préparer les réunions.

3. TODO

Le chef de projet est responsable de toujours fournir une liste de todo à jour aux développeurs.
Dans cette liste doit apparaître :

- les scénarios utilisateur (notion XP)
- tâche de projet
- rapport de bug détaillé (avec la démarche pour reproduire le problème)
- les autres idées ou remarques.

à charge des développeurs :

- Passer régulièrement prendre connaissance de cette liste.
- Quand un développeur a terminé un point, il grise la ligne pour le signaler. Il rajoute à la fin de la ligne son nom, pour qu'on sache qui a fait quoi (pour la traçabilité).

à charge du chef de projet :

- trier les points par priorité (urgent, très important, important, moins important).
- ranger les points terminés (grisés) dans une catégorie en fin de document.

L'idée de ce document 'todo' est inspirée des 'tickets' de trac.

3. Règles techniques

0. Méthode de conception

L'architecture globale de l'application a été pensée au début du projet, avec les scénarios utilisateur dont nous disposions. Ensuite, au fur et à mesure des nouveaux scénarios utilisateur, nous avons fait évoluer le modèle pour l'adapter aux nouveaux besoins.
Notre choix a été de décomposer les éléments en module, qui correspondent à des packages de code. Les modules sont le plus indépendant possible, fournissant des services aux autres modules par l'intermédiaire d'appels des méthodes publiques. Ainsi, le développement des modules est indépendant, ce qui nous permet de travailler indépendamment.
Si le modèle de l'application doit être modifié à cause d'un nouveau scénario utilisateur, le refactoring doit être réalisé en commun, pendant une réunion.

1. Environnement et outils de développement

L'utilisation de l'IDE Eclipse avec les plugins GWT et SVN est obligatoire. Il n'y a cependant aucune limitation au niveau du système d'exploitation et des navigateurs utilisés, mais l'encodage en UTF-8 est requis.
La diversité de plateforme permet aux développeurs de comparer le fonctionnement de l'application dans différents contextes et ainsi de vérifier sa portabilité. (Des tests de portabilité plus poussés ont été réalisés, une compte rendu détaillé est disponible dans la section "Support des navigateurs" du manuel utilisateur.)

2. Bonnes pratiques

- Indentation du code : 4 espaces (pour la lisibilité).
- Encodage des fichiers en UTF-8 (pour la portabilité).
- Bien décrire les arguments des fonctions (pour la reprise par un tiers).
- Prévoir l'exploitation par un outil de documentation automatique (javadoc).
- Éviter d'indiquer le fonctionnement du code dans les commentaires.
- Éviter de paraphraser le code.

3. Convention de nommage

Classes et packages

Le nommage des packages et des classes est libre, mais il faut impérativement que les noms soient explicites, et qu'ils contiennent le nom du composant tel que décrit dans les documents de conception.

Base d'objets persistants

L'application utilise la technologie des objets persistants pour garantir la conservation des données des utilisateurs dans le temps. Cette technologie permet la conservation dans le temps d'instances de classes marquées comme persistantes, qui sont stockées avec leurs arguments dans une base de donnée.
Le nommage des classes persistantes doit se faire de manière à ce que le nom de la classe fasse directement référence au modèle représenté.

Interface

L'interface est construite autour des objets du framework SmartGWT. Ce dernier fournit un nombre important de widgets très haut niveau, gérant nativement une multitude de fonctionnalités.
Le nommage des instances des objets graphiques doit évoquer la fonction du widget.

4. Protocoles de test

Une fois qu'un scénario utilisateur est réalisé, vérifier les tests fonctionnels prévus au départ.

Vérifications complémentaires pour les bugs et problèmes techniques :
- Les débordements de tableau
- Les problèmes d'initialisation (généralement signalés par le compilateur)
- Les problèmes liés aux conversions implicites dans les expressions mathématiques
- Tests en local (en vérifiant le log d'erreur de la console Eclipse)
- Tests en déployé, en vérifiant la consommation CPU des requêtes, et les logs du serveur.

5. Standard et norme de programmation

Le code doit respecter la norme SUN pour le JAVA, à quelque exceptions prêt :
- La limitation de taille pour les lignes est étendue à 160 caractères (au lieu de 80)
- Plus de limite de longueur pour les méthodes

6. Contraintes de conception

Pour toutes les classes, adopter cette structure :
- Les constantes.
- Les attributs de la classe.
- Le / les constructeur(s).
- Les méthodes.
- Les getter / setter des attributs.

- Pour la cohérence graphique du logiciel et une bonne compatibilité des modules, les objets SmartGWT sont à privilégier, même s'ils ont un équivalent hors librairie. En effet, les objets "conteneurs" comme les layout SmartGWT sont conçus pour contenir des objets SmartGWT. Cette règle nous permet une intégration des modules (développés séparément) plus facile.

7. Règles d'utilisation du gestionnaire de version : SVN

0. Commits

Un commit est une publication de sources: c'est une opération de base de SVN. Elle consiste en la mise à disposition aux autres du travail réalisé par un programmeur depuis la dernière version publiée.
Avant de faire un commit, le programmeur doit s'assurer :
- qu'il possède la dernière version des sources (sinon faire un update)
- résoudre les conflits éventuels
- s'assurer du bon fonctionnement de la globalité de l'application

Il est demandé de ne pas publier les répertoires et fichiers suivants :
- le dossier de fichiers compilés (/war/ccalendar/)
- les fichiers de configuration de l'IDE (./classpath)
- le dossier /war/WEB-INF/appengine-generated/

Pour bien respecter ces consignes, il est conseillé de sélectionner manuellement les fichiers/packages à commiter, et non la racine du projet.

1. Update

L'update du projet permet de mettre à jour les fichiers de la copie locale par rapport à la version la plus récente sur le serveur. En cas de conflit, le plugin utilisé dans éclipse indique les zones provoquant le conflit en mettant en évidence notre version et celle du serveur.
On update le projet pour travailler à chaque fois avec la version à jour du projet. L'idéal est de lancer l'update à chaque lancement d'éclipse, et à chaque fois qu'un autre développeur indique avoir comité.

2. Checkout

Le Checkout est l'opération qui consiste à récupérer pour la première fois les fichiers déjà existant au sein d'un projet du dépôt. Cette opération ne se fait en général qu'une fois par projet. Le résultat est une copie de travail.

3. Import

L'import est l'opération inverse du Checkout. Elle consiste à placer dans le dépôt des fichiers locaux déjà existants pour y créer un nouveau projet. Cette opération ne se fait en général qu'une fois par projet.

8. Règles d'utilisation des serveurs d'application google

Google autorise chaque compte Gmail à créer jusqu'à 10 appspots. Chaque développeur peut donc utiliser ces comptes à sa guise pour ses tests en version déployé.
Le compte google "calendriercollaboratif@gmail.com" est propriétaire des appspots utilisés pour les démonstrations au client. Ces appspots sont :

- calendar-alpha.appspot.com
- calendar-beta.appspot.com
- calendier-collaboratif.appspot.com

9. Règles de gestion des erreurs et des exceptions

Les erreurs coté serveur sont notifiées au client par le résultat de l'appel asynchrone.
En cas d'erreur avec le datastore (base d'objets persistants), c'est le serveur Google qui notifie l'erreur dans un log serveur. Les erreurs les plus fréquentes pour notre application sont les erreurs de communication avec le serveur (perte de connexion internet). Elles sont notifiées à l'utilisateur avec l'apparition d'une fenêtre modale, qui décrit l'erreur qui vient de se produire. La navigation est donc interrompue, et l'utilisateur est invité à recharger la page.

4. Communication, charte graphique

0. Convention d'interface

- L'utilisation des widgets du framework SmartGWT est obligatoire pour des raisons de compatibilité entre les widgets conteneurs et les contenus.
- L'intégration des composants principaux doit se faire dans les zones prévues à cet effet :
    - La barre de menu prend toute la largeur de la page et est située en haut.
    - La zone de gauche est réservée aux composants contrôlant le calendrier et donnant des informations sur les actions des collaborateurs.
    - La zone de droite est réservée à la visualisation du calendrier ou du planning.
- Quand il est nécessaire d'interagir avec l'utilisateur, pour que ce dernier puisse ajouter ou modifier des informations, on affiche une fenêtre à l'écran (objet Window de SmartGWT) par dessus l'interface en fond.
- Les styles CSS sont définis dans le fichier "ccalendar.css". Les sections correspondant aux différents modules sont délimitées par des commentaires.
- Pour informer l'utilisateur d'une erreur, on utilise une fenêtre modale contenant un message d'erreur détaillé.

1. Documents livrables

Rédaction sous google doc :

- Sommaire automatique
- polices 'Arial'
- taille de caractères 11
- texte justifié
- couleur noire sur fond plan
- mise en page identique sur tous les documents

2. Communication avec le client

Tous les documents sortant du cercle des développeurs, à destination de l'extérieur, doivent impérativement avoir été relus par le chef de projet.

3. Communication avec les autres groupes

Aucune information précise ne doit être communiqué aux membres des autres groupes. Il faut éviter la "fuite des idées".

D'autre part, il est demandé d'être à l'écoute des groupes concurrents. La prise en compte de leur analyse du problème, potentiellement différente de celle de notre équipe peut permettre une meilleure compréhension de l'attente du client.

5. Fiabilité des serveurs et des librairies

Les serveurs Google étant développés spécialement pour le déploiement d'applications GWT, ils offrent de bonnes performances tout en garantissant une très bonne fiabilité.
Le HDR (High Replication Datastore) fournit un support très fiable pour les données persistantes. Ce système garantit une grande disponibilité des données, et est très sûr. Il résiste par exemple très bien, même en cas de panne du serveur. En contre partie, la propagation des données est plus coûteuse qu'avec un système de stockage classique (master-slave).

De son côté, le framework SmartGWT offre une bonne compatibilité avec les navigateurs récents, ce qui garantit la portabilité de l'application. De plus SmartGWT est activement développée et de nouvelles versions sont régulièrement publiées, ce qui garantit une très bonne fiabilité des composants, et des corrections rapides en cas de bug ou de faille de sécurité.

6. Lexique

Master/Slave : A master-slave replication system asynchronously replicates data as you write it to another physical datacenter. Since only one datacenter is the master for writing at any given time, this option offers strong consistency for all reads and queries, at the cost of periods of temporary unavailability during datacenter issues or moves. Offers the lowest storage and CPU costs for storing data. (source : https://appengine.google.com)

High Replication : Uses a more highly replicated Datastore that makes use of a system based on the Paxos algorithm to synchronously replicate data across multiple locations simultaneously. Offers the highest level of availability for reads and writes, at the cost of higher latency writes, eventual consistency for most queries, and approximately three times the storage and CPU cost of the Master/Slave option. (source : https://appengine.google.com)