10/02/2023

Livrable 1

EasySave

L.Barreto, V.Goulier, A.Martinez, A.Milhas

Table des matières

I - Introduction :

Parties prenantes :

Société ProSoft :

- L.Barreto
- V.Goulier
- A.Martinez
- A.Milhas

Contexte :

Récemment embauchés par l'éditeur de logiciels Prosoft en tant que développeurs, nous avons pour mission de développer un logiciel de sauvegarde, le logiciel EasySave.

Dans ce projet, nous avons la charge de la globalité du projet à savoir : la modélisation du logiciel, le développement de la version initiale et des versions futures, l'édition de release note à chaque mise à jour ainsi que la rédaction de la documentation utilisateur.

Comme tout logiciel de la suite ProSoft, le logiciel EasySave s'intégrera à la politique tarifaire déjà existante.

- Prix unitaire : 200 €HT
- Contrat de maintenance annuel 5/7 8-17h (mises à jour incluses): 12% prix d'achat (Contrat annuel à tacite reconduction avec revalorisation basée sur l'indice SYNTEC)

II – Cahier des charges :

Cahier des charges – Version 1.0 :

Vous trouverez ci-dessous le cahier des charges de la version 1.0 de l'application qui nous a été fournis par la DSI ProSoft :

- Le logiciel est une application Console utilisant .Net Core.
- Le logiciel doit permettre de créer jusqu'à 5 travaux de sauvegarde
- Un travail de sauvegarde est défini par
    - Une appellation
    - Un répertoire source
    - Un répertoire cible
    - Un type (complet, différentiel)
- Le logiciel doit être utilisable à minima par des utilisateurs anglophones et Francophones
- L'utilisateur peut demander l'exécution d'un des travaux de sauvegarde ou l'exécution séquentielle de l'ensemble des travaux.
- Les répertoires (sources et cibles) pourront être sur :
    - Des disques locaux
    - Des disques Externes
    - Des Lecteurs réseaux
- Tous les éléments du répertoire source sont concernés par la sauvegarde
- Fichier Log journalier :

  Le logiciel doit écrire en temps réel dans un fichier log journalier l'historique des actions des travaux de sauvegarde. Les informations minimales attendues sont :
    - Horodatage
    - Appellation du travail de sauvegarde
    - Adresse complète du fichier Source (format UNC)
    - Adresse complète du fichier de destination (format UNC)
    - Taille du fichier
    - Temps de transfert du fichier en ms (négatif si erreur)
    - Exemple de contenu: Sample_log.pdf [pdf]
- Le logiciel doit enregistrer en temps réel, dans un fichier unique, l'état d'avancement des travaux de sauvegarde. Les informations à enregistrer pour chaque travail de sauvegarde sont :
    - Appellation du travail de sauvegarde
    - Horodatage
    - Etat du travail de Sauvegarde (ex : Actif, Non Actif...)

Si le travail est actif :

- Le nombre total de fichiers éligibles
- La taille des fichiers à transférer
- La progression
  - Nombre de fichiers restants
  - Taille des fichiers restants
  - Adresse complète du fichier Source en cours de sauvegarde
  - Adresse complète du fichier de destination
- Exemple de contenu : Sample_state.pdf [pdf]

- Les emplacements des deux fichiers (log journalier et état) devront être étudiés pour fonctionner sur les serveurs des clients. De ce fait, les emplacements du type « c:\temp\ » sont à proscrire.
- Les fichiers (log journalier et état) et les éventuels fichiers de configuration seront au format JSON. Pour permettre une lecture rapide via Notepad, il est nécessaire de mettre des retours à la ligne entre les éléments JSON. Une pagination serait un plus.

Comme énoncé précédemment, ce cahier des charges est celui qui nous a été fournis par notre hiérarchie. Afin de clarifier certains points et pouvoir déceler les points les plus importants, nous avons organisé une réunion afin d'analyser ce cahier des charges et en clarifier le contenu. A la fin de cet échange, nous avons abouti à cela :

Fonction	Expression	Critères	Niveaux	Flexibilité
FP1	Sauvegarder les répertoires indiqués par les utilisateurs.	-Sur un disque local, externe ou réseau. -Tous les éléments du répertoire sont concernés.		1 1
FC1	Doit être une application Console utilisant .Net Core.	-Langage de programmation. -Framework.	C# >= .NET Core 3.X	0 0
FC2	Doit créer jusqu'à 5 travaux de sauvegarde[1].			1
FC3	Doit être utilisable par tous les collaborateurs de l'entreprise.	-Disponible dans d'autres langages.	Au minimum en français et anglais.	0
FC4	Exécuter un ou séquentiellement l'ensemble des travaux de sauvegarde.			0
FC5	Doit enregistrer des logs journaliers des actions des travaux de sauvegarde.	-Contient (« Liste information log[2] »). -Fonctionne sur les serveurs clients. -Format JSON[4].		2 3 2
FC6	Doit enregistrer un fichier d'états d'avancement des travaux de sauvegarde.	-Contient (« Liste information avancement[3] »). -Fonctionne sur les serveurs clients. -Format JSON[4].		2 3 2
FC7	Doit être modifiable en application WPF.	-Architecture	MVC	0
FC8	Doit être facilement maintenable.	- - - Respect des conventions de nommage. - Versioning	 GIT	 0

Le tableau ci-dessus identifie huit fonctions contraintes ainsi qu'une fonction principale pour cette version 1.0 du logiciel EasySave.

III – Diagrammes :

Afin de gagner encore en efficacité lors du développement de l'application mais aussi pour clarifier le cahier des charges au niveau logiciel, nous avons édité plusieurs diagrammes qui sont exposés ci-dessous.

Diagramme Use Case :

Ce diagramme permet de définir les cibles de l'application EasySave.

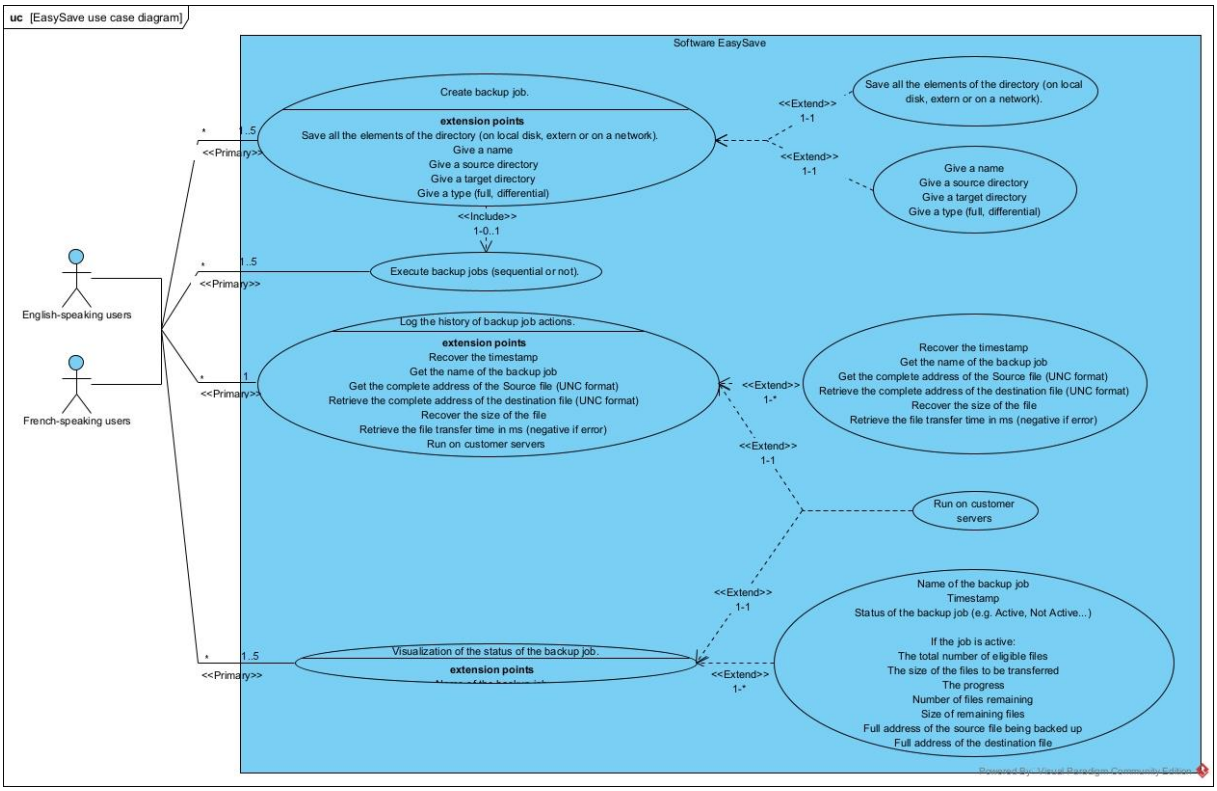


Diagramme use case EasySave

Diagramme d'activité :

Ce diagramme permet de recenser les actions que l'utilisateur pourra effectuer lors de l'utilisation de l'application.

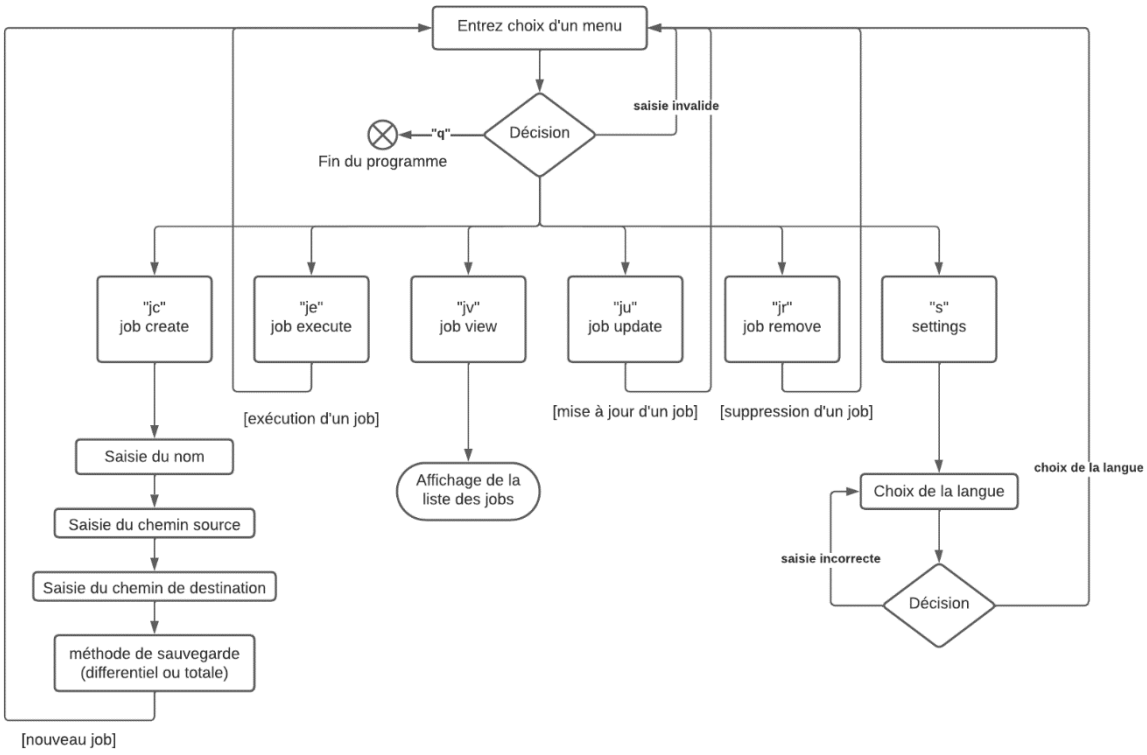


Diagramme d'activité EasySave

Diagramme de classe :

Ci-dessous sont présents dans l'ordre les diagrammes de classe de EasySaveLib ainsi qu'EasySaveConsole. Ces diagrammes détaillent notre architecture de part les différentes classes présentes ainsi que les différents liens entre elles. Les diagrammes sont disponibles en annexes.

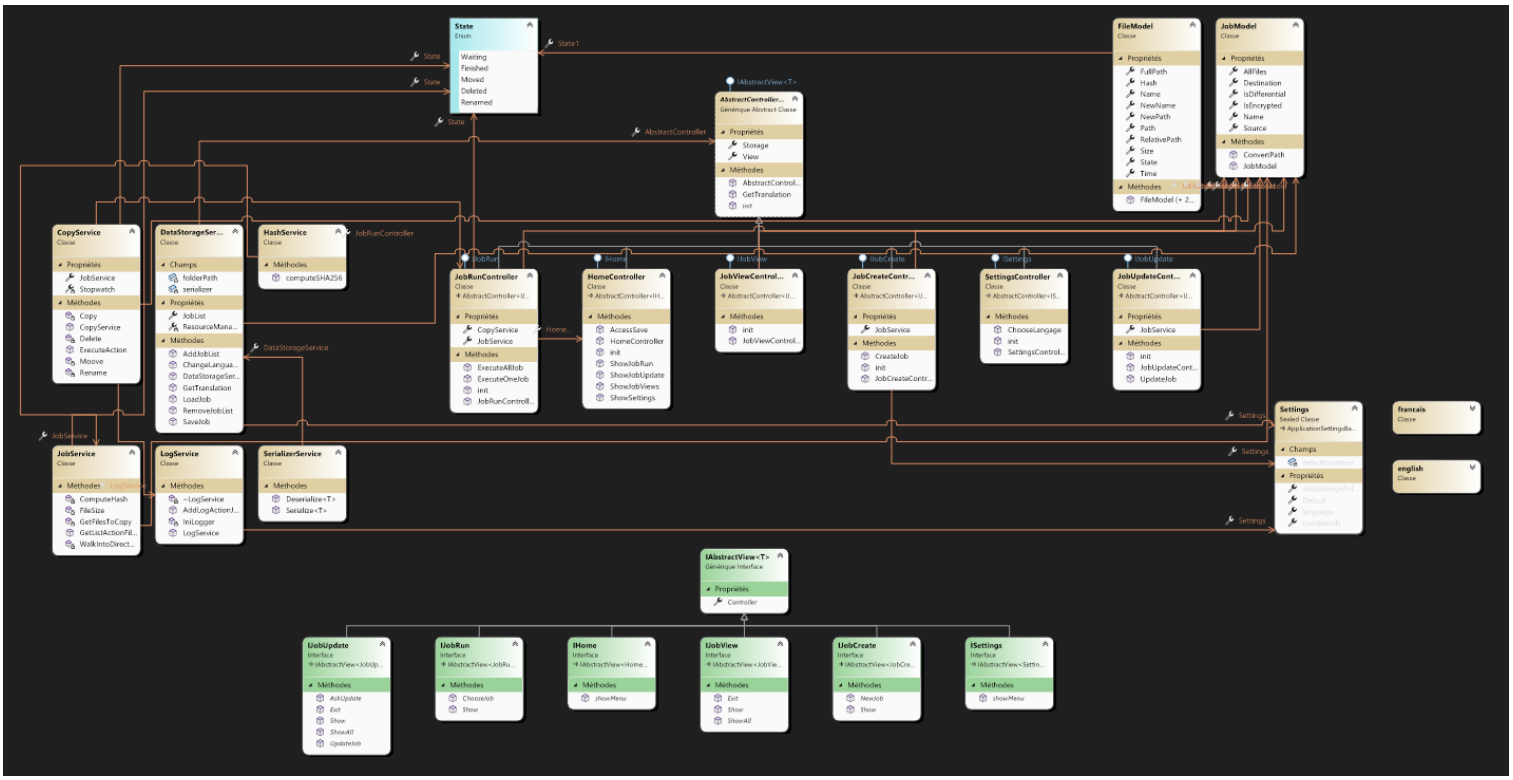


Diagramme de classe EasySaveLib

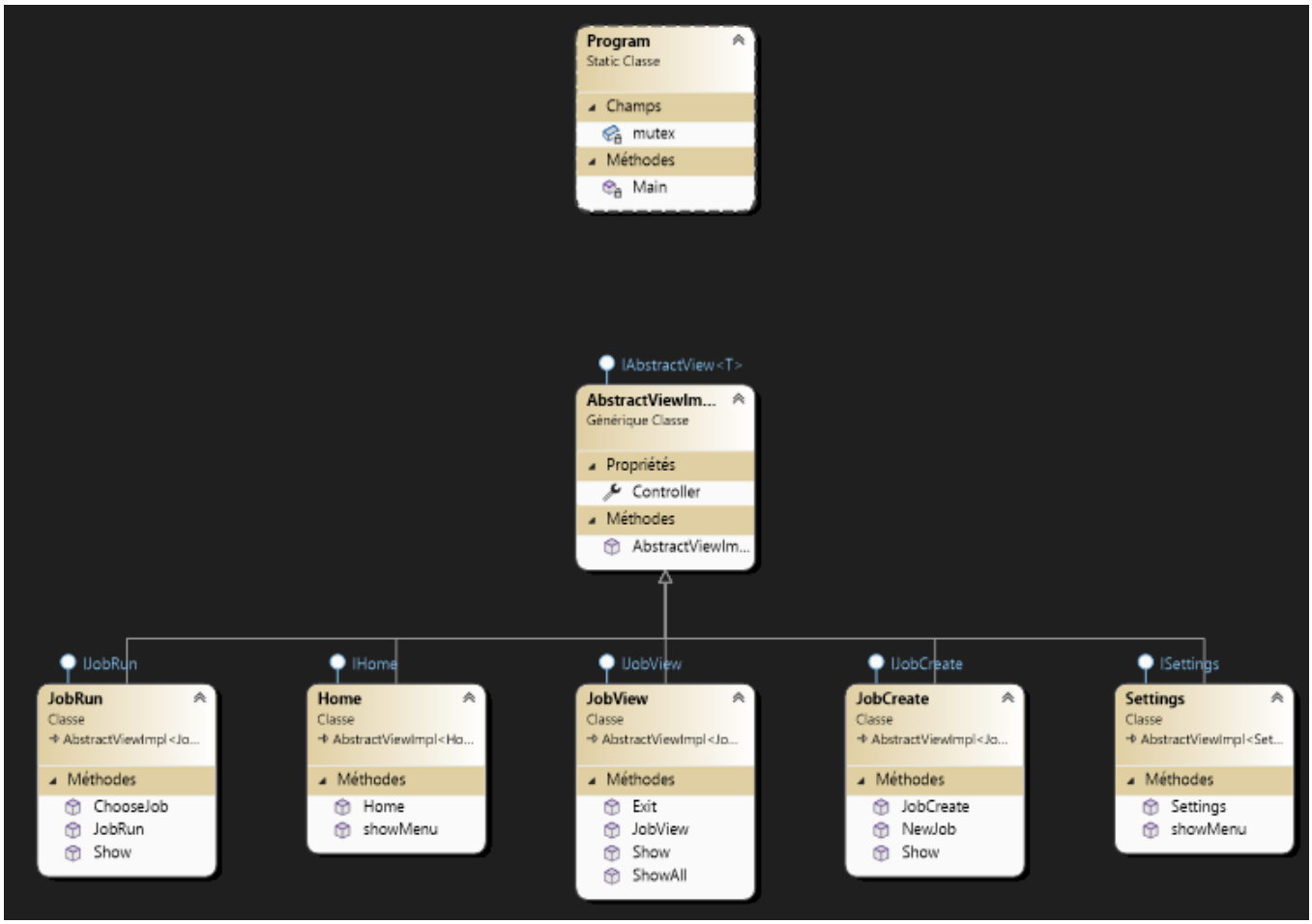


Diagramme de classe EasySaveConsole

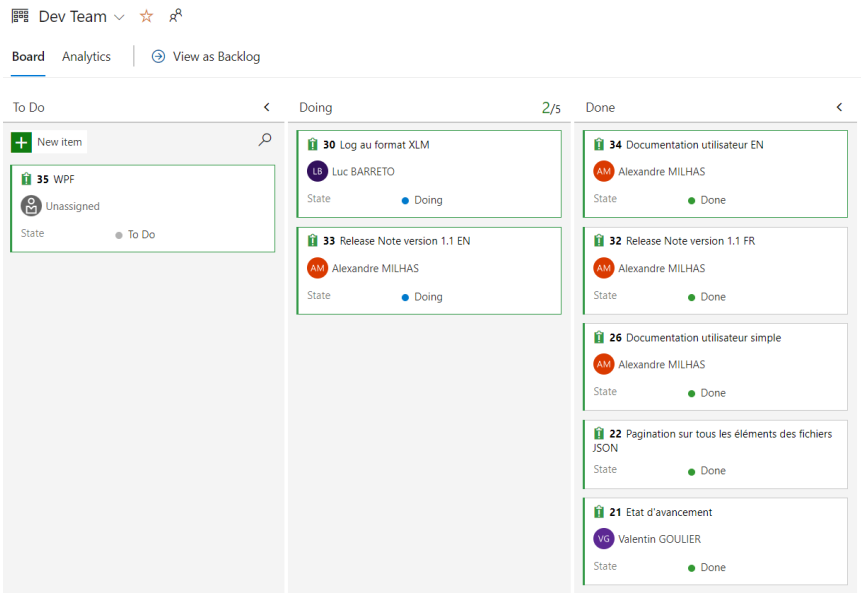
Diagramme de Séquence :

Nos équipes sont actuellement en train de travailler sur ce diagramme, il sera disponible dans les ressources de notre Intranet.

IIII – Commentaires :

Ici, nous détaillerons nos différents choix techniques et explications à propos de la version 1.0 du logiciel EasySave.

Tout d'abord, nous utilisons l'outil Azure DevOps à plusieurs niveaux. En effet, l'outil intègre le « Board », un espace d'attribution de tâches qui nous permet de répartir les différentes tâches entre les différents développeurs. Nous pouvons suivre l'avancement et le déroulement de ces tâches n'importe quand et en temps réel. En parallèle de cela, Azure DevOps intègre Git. Cela nous permet de regrouper le travail de chacun mais aussi d'assurer une fonction de versionning. Nous sommes donc capables de retracer les différentes évolutions du code de notre application. Finalement, nous effectuons des tests automatisés grâce à Azure DevOps.



En ce qui concerne notre framework, nous avons décidé d'utiliser .Net Core 6 pour la réalisation de nos applications console. Nous partons sur cette solution pour plusieurs raisons : version récente, bonne pérennité, bien optimisé et complet. De plus, cette solution est bien indiquée pour l'usage en multi plateformes. Pour compléter, nous utilisons le langage C# (CSharp).

Nous avons reçu de notre hiérarchie la consigne de travailler en pensant à la réduction des coûts de développement des versions futures de l'application. C'est pour cela, que nous avons opté pour une architecture MVC (Model, Vue, Controller – Modèle, Vue, Contrôleur). Le choix de cette architecture nous permet de conserver le squelette de l'application tout en laissant ouverte la possibilité de modifier ou d'ajouter des fonctionnalités à l'application. Le développement d'une base qui restera inchangée dans le futur permet de se garantir une base qui n'aura pas de besoin de redéveloppement et donc des coûts seulement lors de la première itération de l'application.

Dans le cadre du support technique de l'application la gestion des logs est cruciale. Pour gérer ces logs, nous avons utilisé la librairie : SerieLog.
SerieLog est un 'logger' qui permet d'éditer des fichiers de logs tout en étant threadsafe et en permettant l'édition de fichers dans plusieurs formats (JSON, XML).
Dans le cadre de notre édition de logs, on récupère l'horodatage, l'appellation du Job, l'adresse du fichier source et du fichier de destination, la taille du fichier et son temps de transfert.
Dans les faits, SerieLog créé un dossier Log qui contient tout l'historique des actions des travaux de sauvegarde de façon journalière. Le chemin du dossier est au même endroit que le Job.JSON qui est le fichier qui répertorie tous les Job de l'utilisateur. Bien sûr, ce chemin est paramétrable pour s'adapter aux volontés clients.
Finalement, la librairie nous permet de changer de fichier tous les jours et permettre une gestion journalière simple. A partir du moment où le fichier dépasse un nombre de lignes donné, un nouveau fichier va être automatiquement créé.

À la vue de la portée voulue internationale de l'application, nous devons proposer une version multilingue de celle-ci. C'est dans ce but que nous utilisons la classe CultureInfo.
Pour la traduction, nous utilisons donc la classe CultureInfo de l'espace de noms System.Globalization. Cette classe permet de définir des informations sur une culture afin de pouvoir adapter une application à une culture.
Plus précisément, nous avons créé des fichiers '.resx' qui sont des fichiers ressources. Dans ces fichiers ressources sont présents des équivalences entre des une variable de type string et une valeur de texte. Le fonctionnement est le suivant : nous disposons d'un fichier ressource par langue. Sur chaque fichier ressource, les variables sont les mêmes mais renvoient des valeurs différentes. Ainsi, il nous suffit d'implémenter une fonction qui permet de sélectionner la langue (et donc la ressource associée) à utiliser et la variable va renvoyer la valeur correspondante dans la bonne langue.
En conséquence de l'utilisation de cette classe, nous n'avons pas de texte en dur dans notre code.

Conclusion :

Ce diagramme résume donc nos activités d'élaboration de l'application, depuis la création de l'architecture, jusqu'au développement en détaillant nos solutions techniques. La notice utilisateur ainsi que les release notes sont disponibles en annexe.